Today's Content:

→ N party ✓

→ Min no of perfect squares to get sum = k ✓

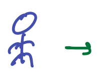
→ Max Subsequence Sum ✓

→ Basic 2D matrix Problems ✓

4Q) Given N persons, How many ways we can pair all people

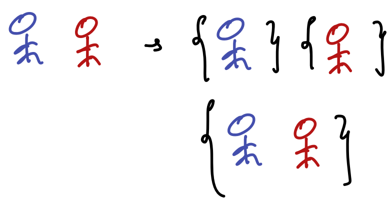
Note: A person either wants to stay alone or get paired

N=1: ways



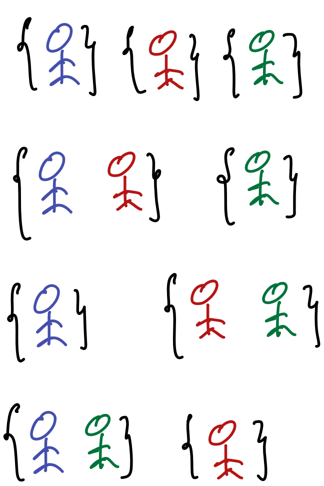
1

N=2: ways: 2



N=3:

ways: 4



eg: $P_1 \; P_2 \; P_3 \; P_4 \; P_5 \; P_6 \; P_7$

{7 people}

Alone → {6 people}

Paired → {5 people}

{6 people}: Alone → {5 people}, Paired → {4 people}

{5 people}: Alone → {4 people}, Paired → {3 People}

80%

// p[i] = Number of ways i people can pair?

i → i people

Dp expression: dp[i] = dp[i-1] + (i-1) * dp[i-2]

1 2 3 4 5 . . i-2 i-1 i

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

i^th person alone

	paired	people
i^th →	1	: dp[i-2]
i^th →	2	: dp[i-2]
i^th →	3	: dp[i-2]
⋮		
i^th →	i-1	: dp[i-2]

dp[i] = dp[i-1] + (i-1) * dp[i-2]

// dp[N+1].

// $dp[i] = dp[i-1] + (i-1) * dp[i-2]$ Base conditions

$i=1$ $dp[1] = dp[0] + (0)(dp[-1])$: $dp[1] = 1$

$i=2$ $dp[2] = dp[1] + (1)\, dp[0]$: $dp[2] = 2$

or $dp[0] = 0$, $dp[1] = 1$

// $dp[1] = 1$, $dp[2] = 2$

```
i = 3; i <= N; i++) {
    dp[i] = dp[i-1] + (i-1)*dp[i-2]
}
```

// return dp[N]

TC: (N states) * O(1)

SC: (N) $\xrightarrow{\text{we are atmax}}$ { 3 variables } → {TODO}

depending on 3 states

SC: O(1)

4Q) Find min no of perfect squares needed to get sum = N

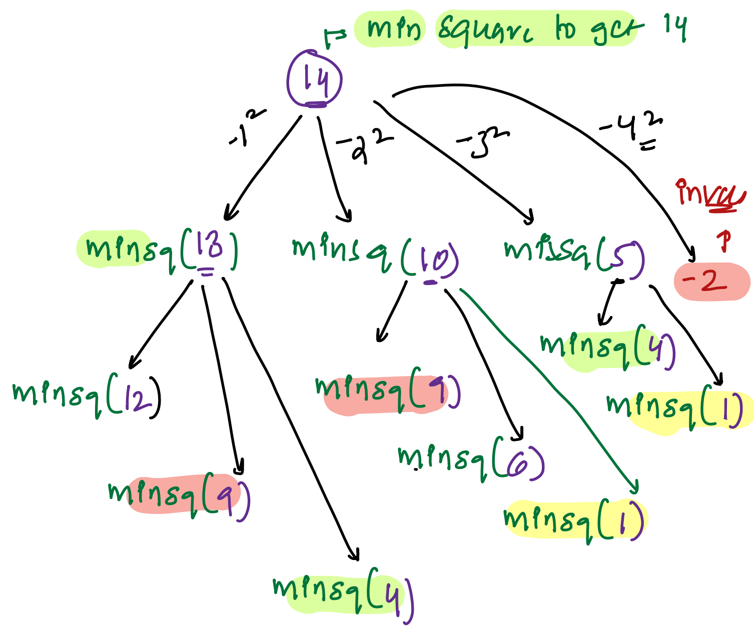
$N=6: \rightarrow \{2^2 + 1^2 + 1^2\}$

$N=10: \rightarrow \{3^2 + 1^2\}$

$N=9: \rightarrow \{3^2\}$

$N=12 \rightarrow \begin{cases} 3^2 + 1^2 + 1^2 + 1^2 \\ 2^2 + 2^2 + 2^2 : 3 \end{cases}$

Idea: greedy won't work



// dp state: dp[i] = min square to get i

min square to get $i-1$

$$// \text{ dp Expression } \quad dp[i] = \min \begin{cases} dp[i-1] + 1 \\ dp[i-2^2] + 1 \\ dp[i-3^2] + 1 \\ dp[i-j^2] + 1 \\ i >= j^2 \end{cases}$$

// dp Table [N+1]

// Base Case :

dp[0] = 0

i = 1 ; i <= N ; i++) {

ans = ~~i~~ INT_MAX / N / N+1

j = 1 ; j * j <= i ; j++) {

ans = min (ans, dp[i - j^2] + 1

}

dp[i] = ans

}

return dp[N]

TC : # states * (TC for each state)

states → N, TC for each state → √N

{10:35 pm}

$\boxed{TC: N\sqrt{N}}$

$\boxed{SC: O(N)}$ → not possible : { Because we are depending on continuous states }

// N = 15

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp[16] :	0	1	2	3	1	2	3	4	2	1	2	..	..	..	..	..

dp[1] = dp[1 - 1²] + 1

dp[2] = dp[1] + 1

dp[3] = dp[2] + 1

dp[4] → min { dp[3] + 1, dp[0] + 1 }

dp[5] → dp[4] + 1, → dp[1] + 1

dp[6] → dp[5] + 1, → dp[2] + 1

dp[7] → dp[6] + 1, → dp[3] + 1

dp[8] → dp[7] + 1, → dp[4] + 1

{We can pick any element}

3Q) Given N ar[] elements find max subsequence sum :

	0	1	2	3	4	5
Ex1:	2	-4	5	3	-8	1

} idea : pick all +ve elements

Ex2: -4 -2 -3 -10 : } Edge case: If all are negative pick max element

4Q) Given N ar[] elements find max subsequence sum :

Note: In a subsequence 2 adjacent elements cannot be present

					ans
Ex1:	9	14	3	:	14
Ex2:	9	4	13	24 :	33
Ex3:	13	14	2	:	15

	0	1	2	3	4	5	6	7
Ex: ar[8] =	2	-1	-4	5	3	-1	4	2

1) optimal substructure ✓

2) overlapping subproblems ✓

ms[0-7]

leave / pick it

max (ms[0-6] , ar[7] + ms[0-5])

leave / pick — leave / pick it

max (ms[0-5] , ar[6] + ms[0-4]) (ms[0-4] , ar[5] + ms[0-3])

dp state: dp[i] = max subsequence sum with no adjacent from [0, i]

0 1 2 3 4 . . i-2 i-1 (i)

fail

dp exp: dp[i] = max(dp[i-1], ar[i] + dp[i-2])

fail, i=0, i=1

dp table: dp[N]

Base: dp[0] = A[0], dp[1] = max(ar[0], ar[1])

Code:

```
i = 2; i < N; i++) {
        dp[i] = max( dp[i-1], ar[i] + dp[i-2] )
}
```

(i, i-1, i-2)

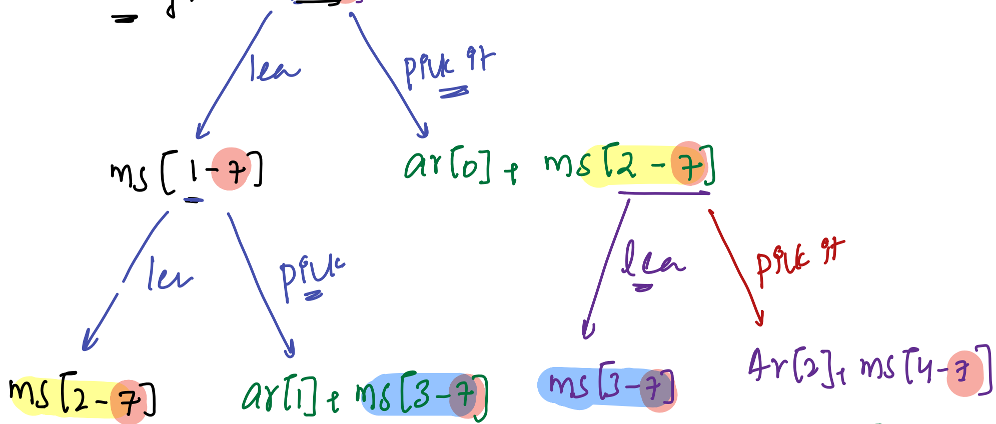
return dp[N-1]

TC: O(N) SC: O(N) $\xrightarrow{\text{3 states}}$ O(1)

Ex: ar[8] =

	0	1	2	3	4	5	6	7
ar[8] =	2	-1	-4	5	3	-1	4	2
dp[8] =	2	2	2	7	7	7	11	11

2nd way : ms[0 - 7]



dp[i] = max subsequence sum without adjacent sum (i, N-1)

i   i+1   i+2 - - -   - -   N-1

$$dp[i] = \max(dp[i+1],\ ar[i] + dp[i+2])$$

N-2   N-1

dp table : dp[N]

Base conditions : $dp[N-1] = ar[N-1]$, $dp[N-2] = \max(ar[N-2], ar[N-1])$

N=10

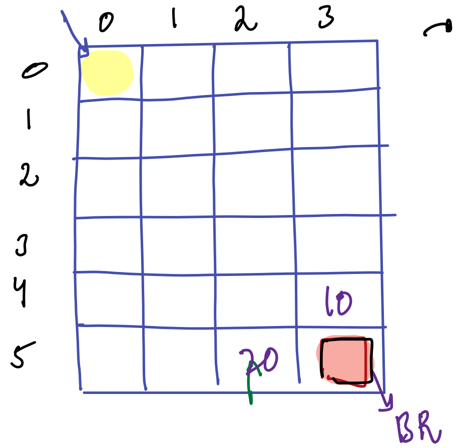
```
i = N-3 ; i >= 0 ; i--) {
    dp[i] = max (dp[i+1], ar[i] + dp[i+2])
}
return dp[0]
```

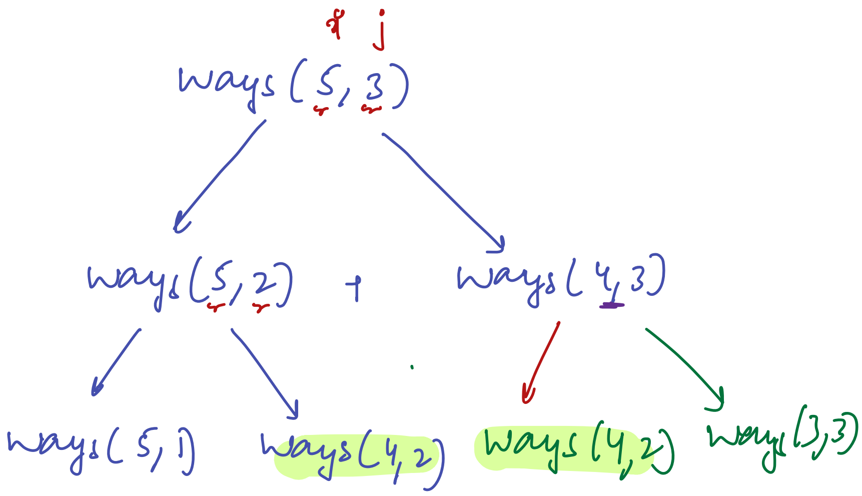
i → i+1 → i+2 → i+3 → i+4

Q8): Number of ways to go from (0, 0) → (BR cell)

From cell → right

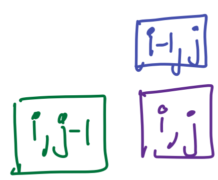
From cell ↓ Bottom





$dp[i][j] = \{$ number of ways to reach $(0,0) \rightarrow (i,j)$ $\}$

$$dp[i][j] = \{ dp[i][j-1] + dp[i-1][j] \}$$

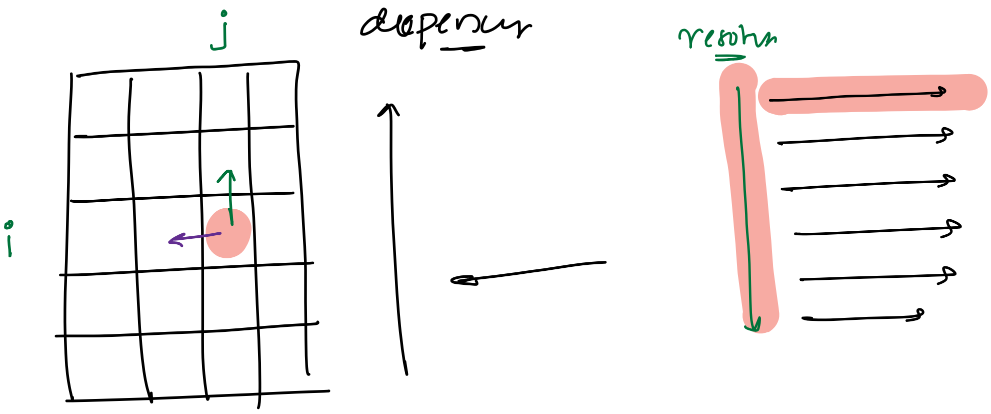


// dp table : $dp[N][M]$

Base Conditions: $\{ i == 0 \; || \; j == 0 \}$ Base condition

$$\forall_{i=0}^{N-1} \; dp[i][0] = 1 \qquad \forall_{j=0}^{M-1} \; dp[0][j] = 1$$

Code:



Pseudocode:

```
i = 1; i < N; i = i+1){
    j = 1; j < m; j = j+1){
        dp[i][j] = { dp[i][j-1] + dp[i-1][j] }
    }
}
```

return: dp[N-1][M-1]

TC: $O(N*M)$ SC: $O(N*M)$ → Think of optimization

2Q) Number of ways to go from (0, 0) → (BR cell)

TODO

Assignment

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

a) From cell → right

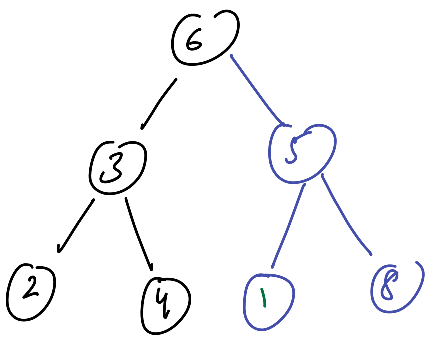
Bottom

b) '0' indicates blocked cell

We cannot go from Blocked cell

Doubts:

K = 12



//