EDIT DISTANCE

Convert A to B.

Operations Permitted:
1) Insert
2) Delete
3) Replace

A = "abad"   B = "abac"

↓ Replace

"abac"

⇒ O/P = 1

A = "Anshuman"
B = "Antihuman"

Replace 's' with 't'

"Anthuman"

↓ Insert 'i'

"Antihuman"

⇒ O/P = 2

A = "Anshuman" (0 1 2 3 4 5 6 7)
B = "Antihuman" (0 1 2 3 4 5 6 7 8)

A = "abcd"
B = "abd"

CASE-I: Insertion: 1 + get_min_steps ("shuman", "ihuman")
= 1 + get_min_steps (A[2...7], B[3...7])
A[i...], B[j+1...]

CASE-II: Deletion: 1 + get_min_steps ("human", "tihuman")
= 1 + get_min_steps (A[3...7], B[2...8])
A[i+1...], B[j...]

CASE-III: Replace: 1 + get_min_steps ("human", "ihuman")
= 1 + get_min_steps (A[3...7], B[3...8])
A[i+1...], B[j+1...]

"abc"
"abd"

if (i == A.length()) ⇒

A = ""   B = "abc"

A = "abc" (0 1 2)
B = "abcde" (0 1 2 3 4)

i = 3
j = 3
5 - j = 5 - 3 = 2

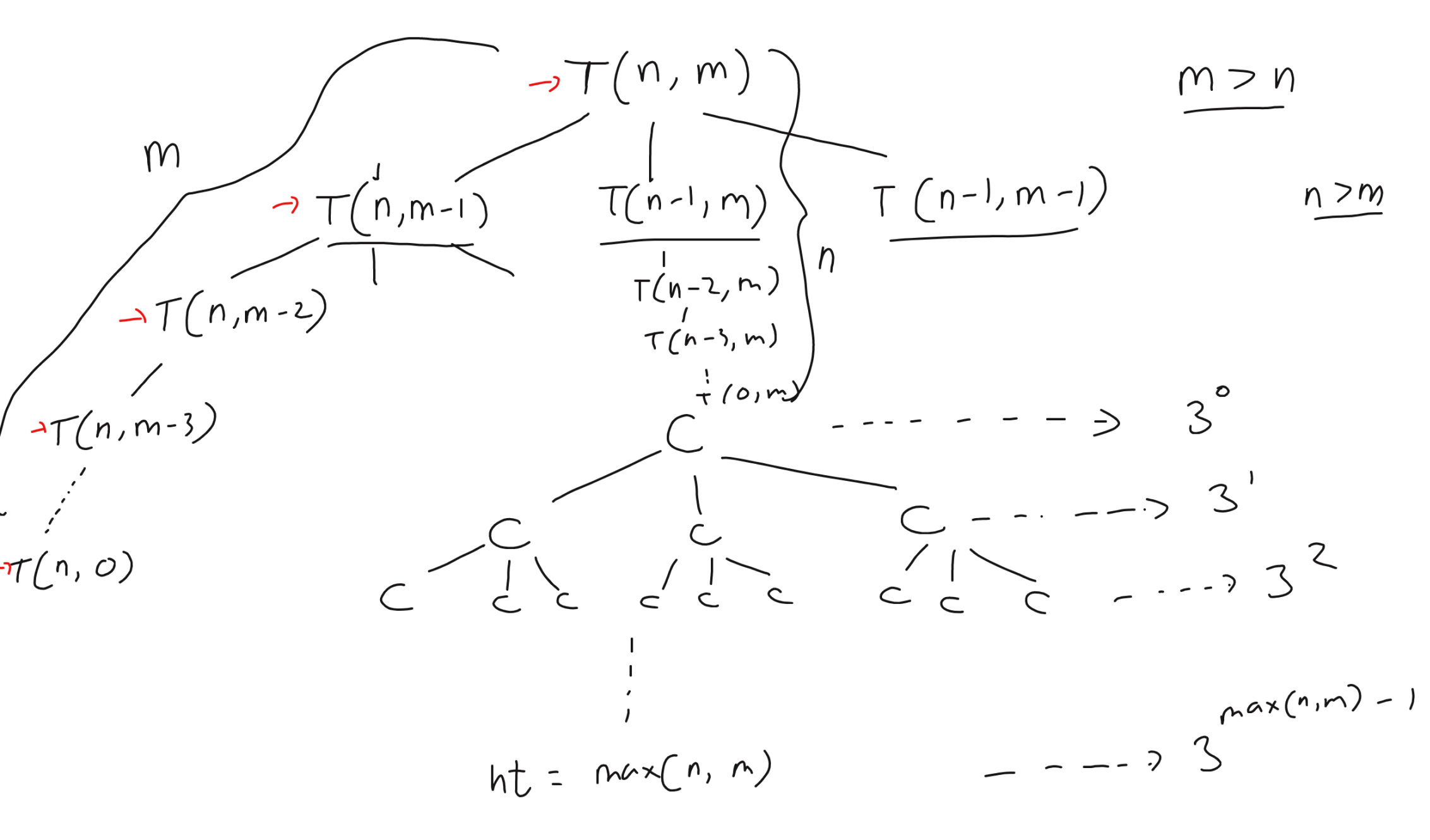
A = "abc"
B = "abc"

i = 3
j = 3
3 - j = 3 - 3 = 0

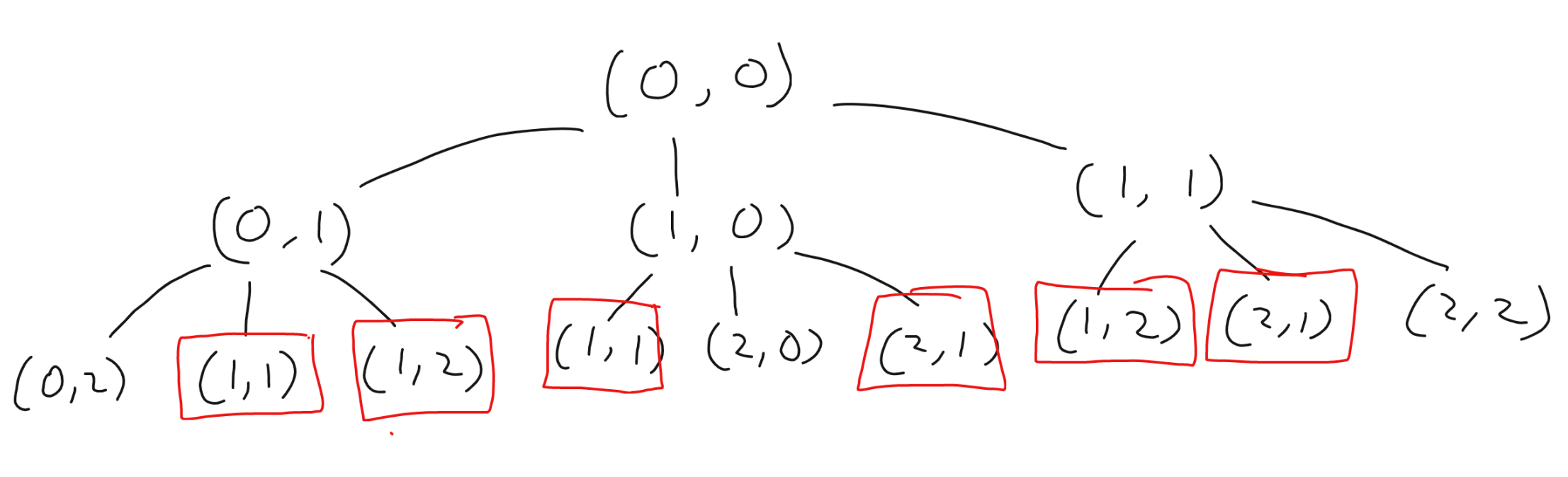
A = "abcde" (0 1 2 3 4)
B = "abc" (0 1 2)

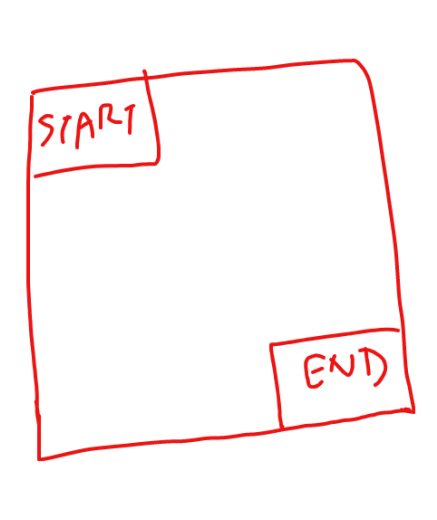
i = 3
j = 3

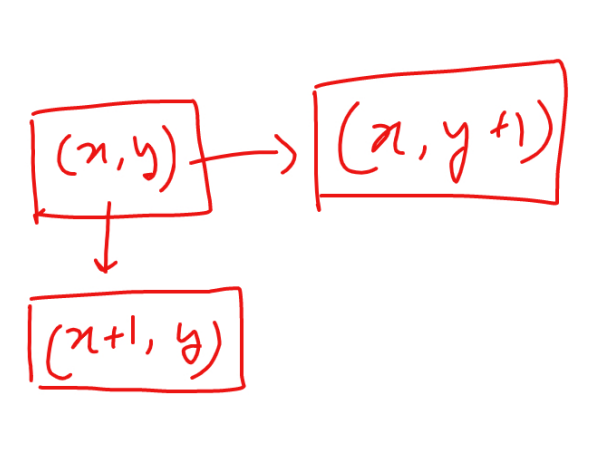
= A.length() - i
= 5 - 3 = 2


T(n, m)
T(n, m-1)
T(n-1, m)
T(n-1, m-1)
T(n, m-2)
T(n, m-3)
T(n, 0)
T(n-2, m)
T(n-3, m)
T(0, m)
m
n
m > n
n > m
C
3^0
3^1
3^2
ht = max(n, m)
$3^{\max(n,m)-1}$

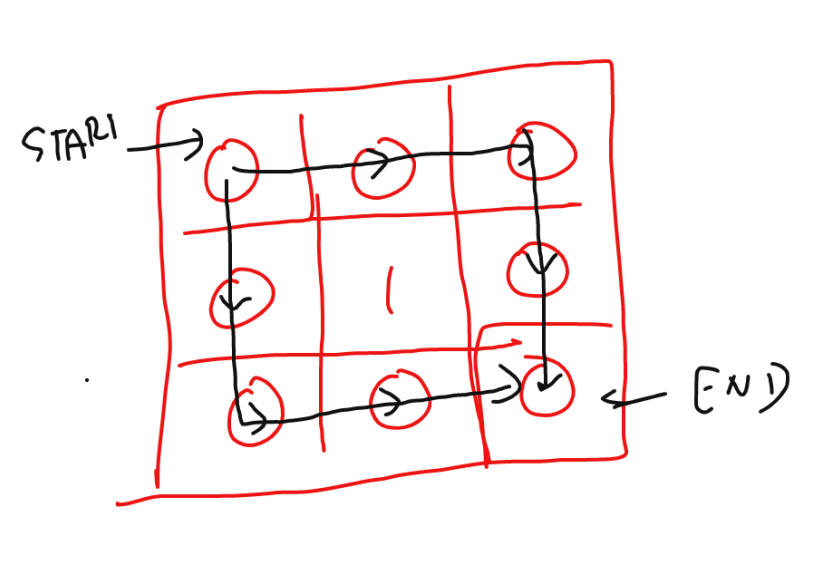
$\Rightarrow O(3^{\max(n,m)})$


(0, 0)
(0, 1)
(1, 0)
(1, 1)
(0, 2)
(1, 1)
(1, 2)
(1, 1)
(2, 0)
(2, 1)
(1, 2)
(2, 1)
(2, 2)

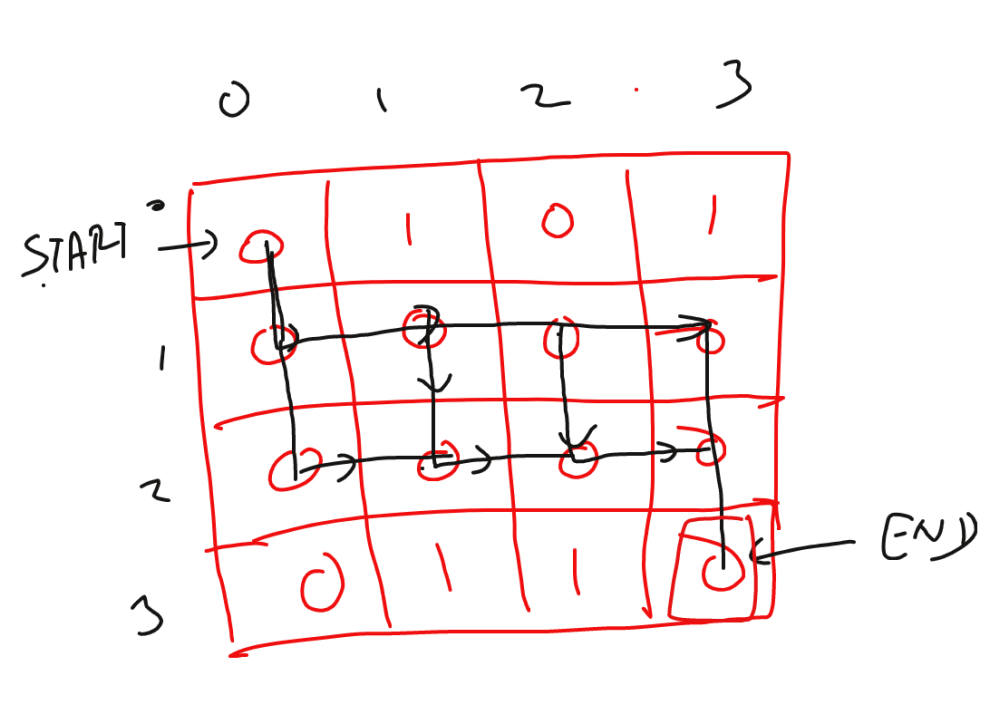




0 → Empty space
1 → Obstacle



⇒ O/P = 2


0
1
2
3
START
1
0
1
1
2
3
0
1
1
END

(0,0) → (0,1) → (1,1) → (1,2)
↓
(1,3)
↓
(2,3)
(3,3)

⇒ O/P = 3

path_count_starting_from (0, 0) = path_count_starting_from (0, 1) + path_count_starting_from (1, 0)
(i, j)   (i, j+1)   (i+1, j)

⇒ BASE-CASE ⇒ if (i == n-1 && j == m-1)
⇒ return 1

(0,0) - (1,0)

mat

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

dp

	0	1	2	3
0	1	0	0	0
1	1	1	1	1
2	1	2	3	4
3	1	0	0	4 → O/P

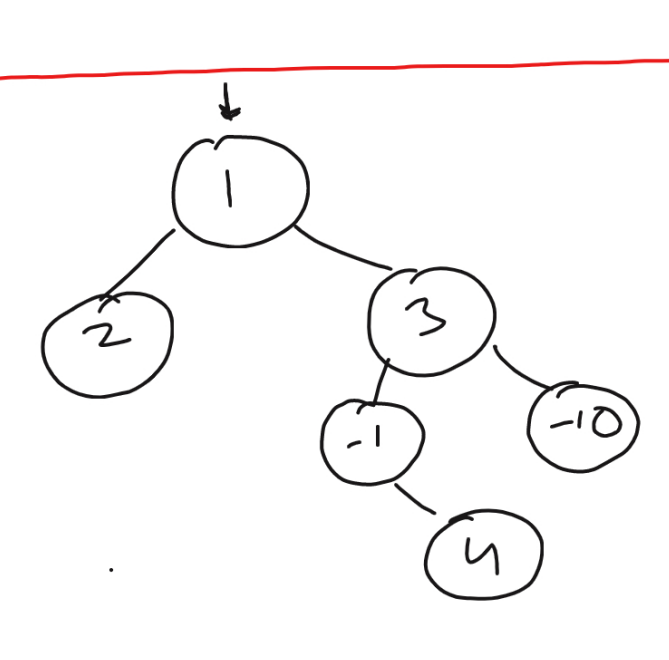
dp[i][j] == no. of paths possible to reach (i, j) from (0, 0)

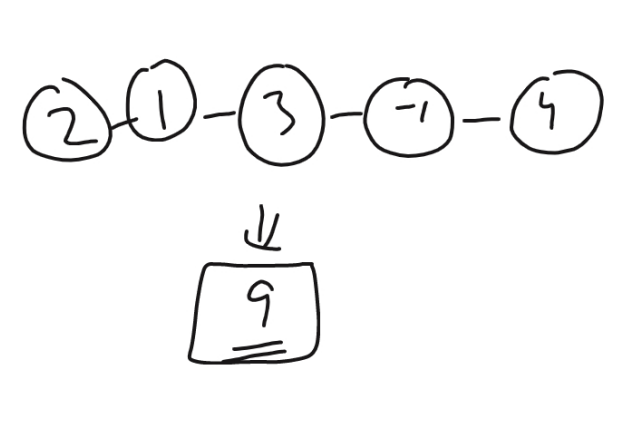
$\Rightarrow \boxed{dp[i][j] = dp[i-1][j] + dp[i][j-1]}$ ⇐

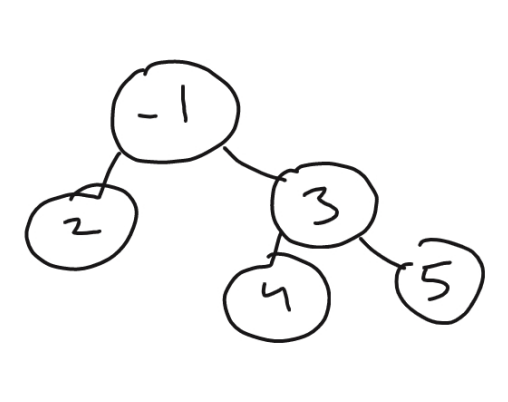
iff (i, j) is not an obstacle

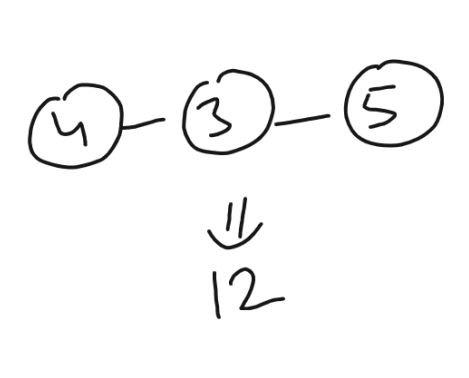
otherwise, if (i, j) is an obstacle: dp[i][j] = 0

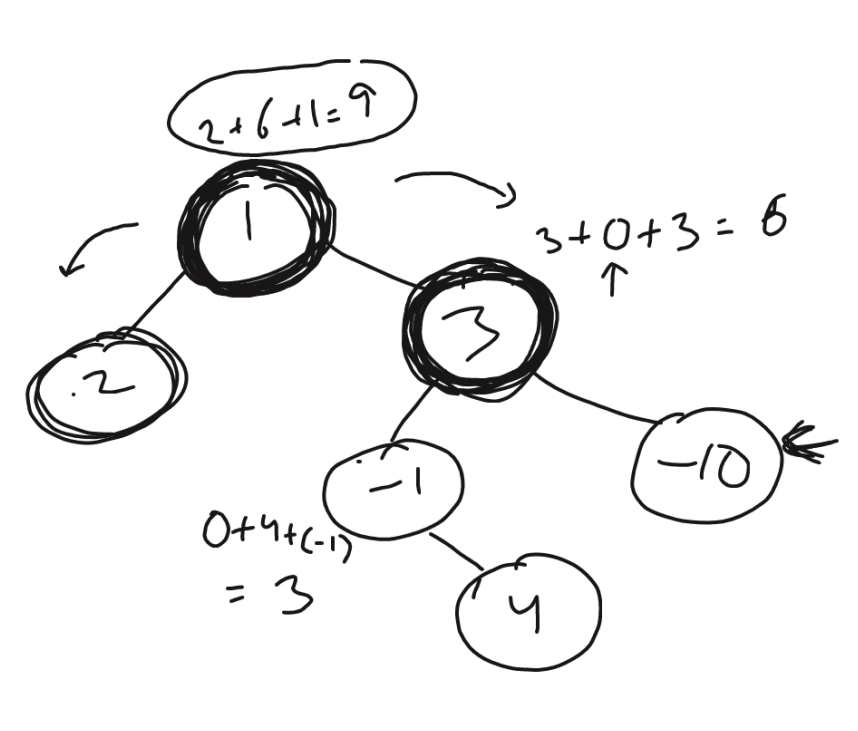
mat ⇒ 0 , 0 , 1 , 0 , 1
dp ⇒ ~~1~~ , ~~1~~ , 0 , 0 , 0 ,





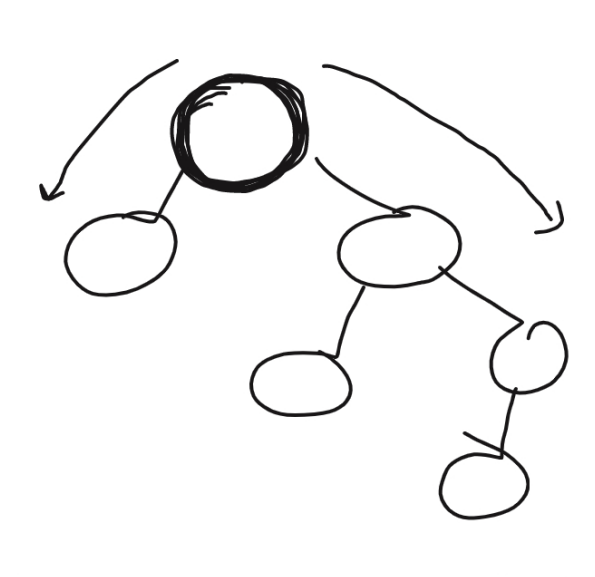





2+6+1=9
3+0+3=6
0+4+(-1) = 3
1
2
3
-1
-10
4

current = 1
→ l_sum = 2
→ r_sum = 6
⇒ sum = l_sum + r_sum + current.val
= 2 + 6 + 1 = 9

For the left & right subtree, instead of heights, we need the maximum sum of any linear path starting from (current → left) & (current → right)



lps(current) = max(current → value, lps(cur → left) + cur → val, lps(cur → right) + cur → val)

OPTIMAL GAME STRATEGY

A: [5, 3, 7, 10]

O/P = 15

I → 10
II → 7
I → 5
II → 3
→ 10 + 5 = 15

A = [8, 15, 3, 7]

O/P = 22

I → 7
II → 8
I → 15
II → 3
→ 7 + 15 = 22

max_score (i, j)

i   j
[8, 15, 3, 7]

I → A[i]
II → A[i+1] or A[j]
I → A[i+2 ... j]   I → A[i+1 ... j-1]

CASE-I
⇒ max_score (i, j) = A[i] + min (max_score (i+2, j), max_score (i+1, j-1)) ①

I → A[j]
II → A[i] or A[j-1]
I → A[i+1 ... j-1]   I → A[i ... j-2]

CASE-II
⇒ max_score (i, j) = A[j] + min (max_score (i+1, j-1), max_score (i, j-2)) ②

---

maximum score possible = max (CASE-I, CASE-II)

$$\Rightarrow \text{max\_score}(i,j) = \max\begin{pmatrix} A[i] + \min(ms(i+2,j), \boxed{ms(i+1,j-1)}), \\ A[j] + \min(\boxed{ms(i+1,j-1)}, ms(i,j-2)) \end{pmatrix}$$

⇒ Dynamic Programming!!