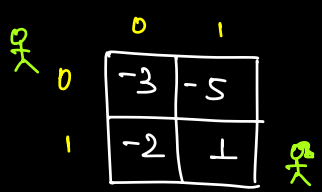
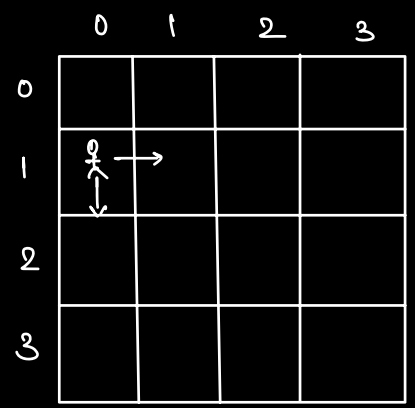
Q.1 Given mat[N][M], where each cell mat[i][j] indicates health gained. Find minimum health required at 0,0 such that we can reach N-1, M-1.

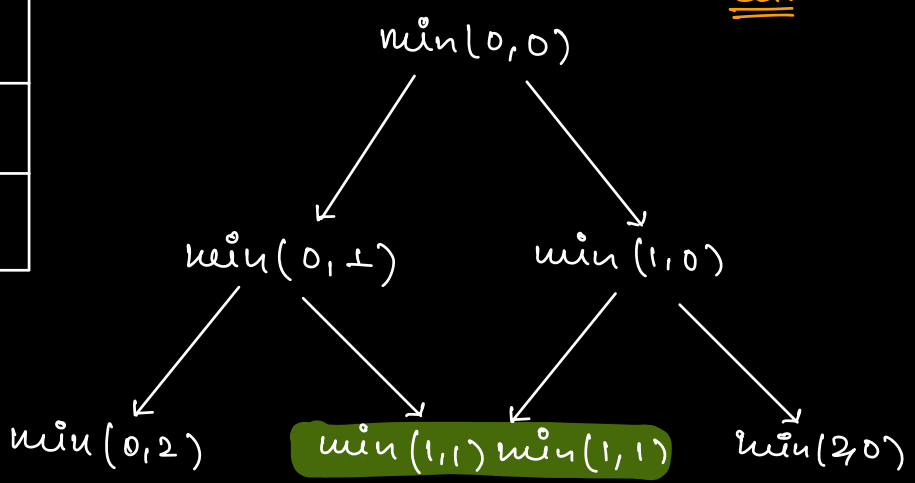
1) Cell → right
   ↓
   down
2) If health reaches zero at any point, that means we are dead.
3) Starting at 0,0.



H = 4 ✗ H = 5 ✗ H = 6



min health required at (0,0) in order to reach BR cell.



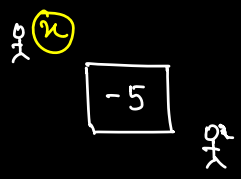
→ Optimal Substructure

→ Overlapping Subproblems.

dp[i, j] = Min. health required at (i, j) s.t we can reach the BR cell.

(N-1, M-1)

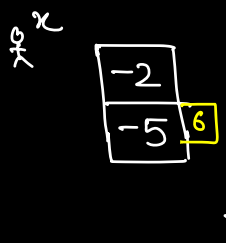
Case I:



$x - 5 = 1$

$x = 6$

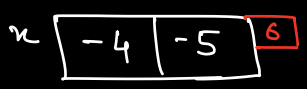
Case II:



$x - 2 = 6$

$x = 8.$

Case III

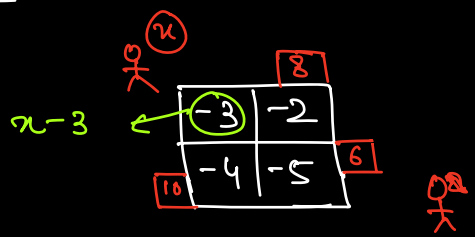


$x + (-4) = 6$

$x - 4 = 6$

$x = 10$

Case IV:

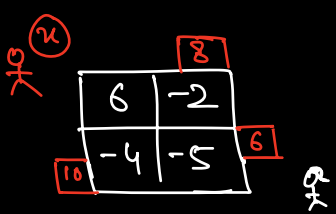


$x + (-3) = \min(8, 10)$

$x - 3 = 8$

$\boxed{x = 11}$

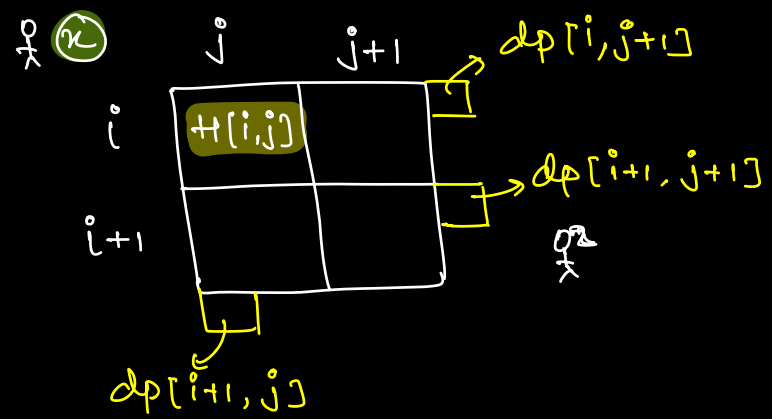
Case V



$x + 6 = \min(8, 10)$

$x + 6 = 8$

$x = 2$



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

$$\boxed{x = \min(dp[i,j+1], dp[i+1,j]) - H[i,j]}$$

x → $dp[i,j]$

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

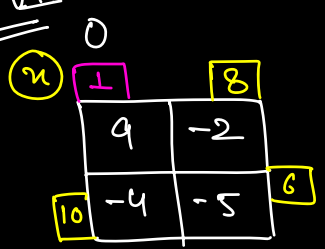
≥ 1

Case I

x → atleast (1)

10

Case II



$x + 9 = \min(8, 10)$

$x + 9 = 8$

$x = -1$ ✗

dp[i,j] = Min. health required at (i,j) s.t we can reach the BR cell.

Dp Expression :

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

Base Case :

i = N-1, j = M-1

dp Table :

int dp[N][M]

Code : Recursive + Memoization.

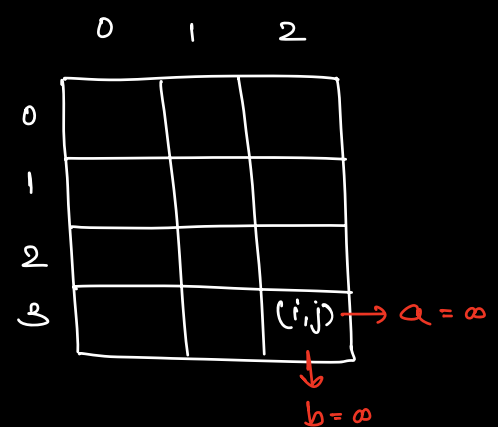
final ans => dp[0,0]

↳ Min Health required at index 0,0 s.t we can reach N-1, M-1

```
int minHealth(int H[][], N, M){
    int dp[N][M] = {-1};
    dp[N-1, M-1] = max(1, 1-H[N-1][M-1])
    fun(dp, H, N, M, 0, 0);
}
```

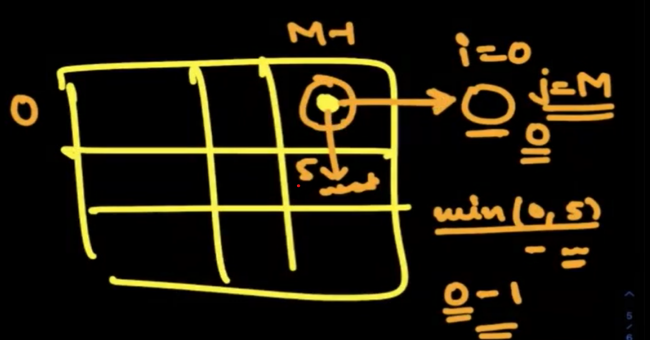
↳ min health required at (0,0)

```
int fun(int dp[][], int H[][], N, M, i, j) {
if(i == N || j == M) {
    return INT_MAX;
}
if (dp[i][j] == -1) {
    a = fun(dp, H, N, M, i, j+1);
    b = fun(dp, H, N, M, i+1, j);
    dp[i,j] = max(1, min(a,b) - H[i,j]);
}
return dp[i][j];
}
```

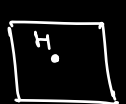


min(∞,∞) ×

Reason why we can't return min value as 0 instead of INT_MAX



(n)



$n + H = 1$

$n = 1 - H$

$n = \max(1, 1-H)$

TC : $O(N \cdot M)$

SC : $O(N \cdot M)$

Q Given 2 strings, find the length of longest common subsequence (LCS)

$S_1 \Rightarrow N$

$S_2 \Rightarrow M$

LCS(S_1, S_2)

S_1 : a b c e f g h

S_2 : a g c h

} ach, agh ⇒ 3

S_1 : a b b c d g g

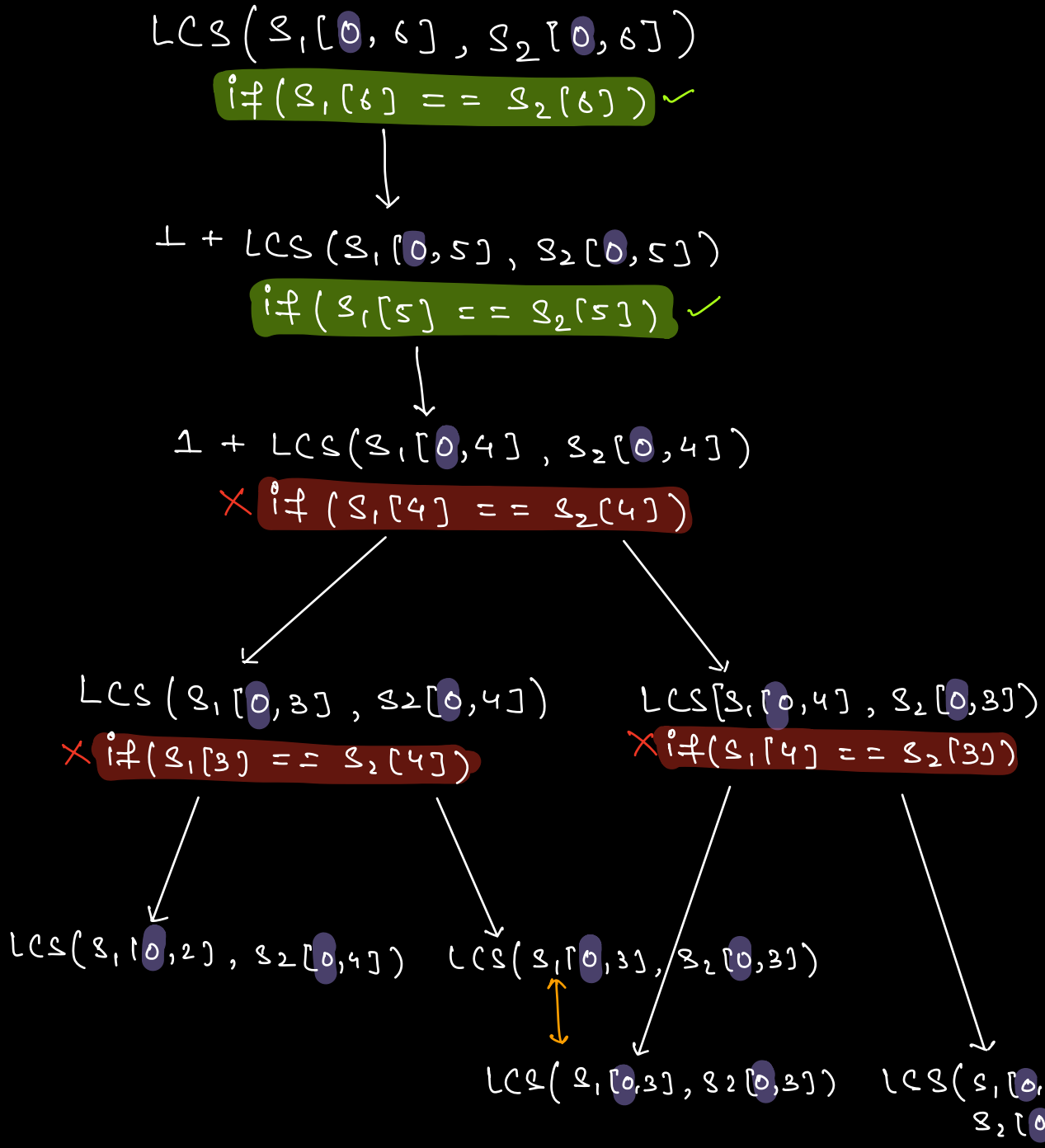
S_2 : b a c h e g f

} acgf, bcgf ⇒ 4

S_1 : K l a g r i p

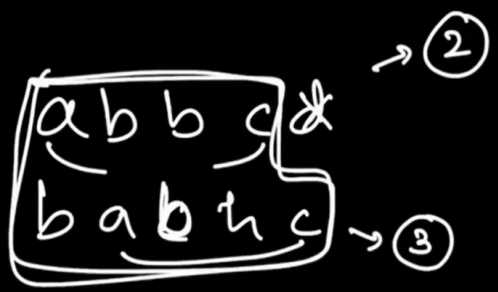
S_2 : l g i g K m

} lgi ⇒ 3



→ Optimal substructure

→ Overlapping subproblems.



dp State

dp[i, j] = LCS of s1[0, i] & s2[0, j]

- i → end index of s1
- j → end index of s2.

dp Expression

$$dp(i,j) = \begin{cases} \text{if } (s_1[i] == s_2[j]) \{ \\ \quad 1 + dp[i-1, j-1] \\ \} \\ \text{else } \{ \\ \quad \max(dp[i, j-1], dp[i-1, j]) \\ \} \end{cases}$$

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

↳ LCS of s1[0, N-1] & s2[0, M-1].

S_1: a (0) ⇒ S_1 : [0, -1]

S_2: a (0) b (1) ⇒ S_2 : [0-0]

int dp[N][M] = {-1}

```
int LCS(String s1, String s2, int i, int j, int dp[][]) {
    if (i == -1 || j == -1) {
        return 0;
    }
    if (dp[i][j] == -1) {
        if (s1[i] == s2[j]) {
            dp[i,j] = 1 + LCS(s1, s2, i-1, j-1, dp);
        }
        else
            dp[i,j] = max { LCS(s1, s2, i-1, j, dp)
                          { LCS(s1, s2, i, j-1, dp) }
    }
    return dp[i,j];
}
```

i = -1
⇒ s1 is Empty.
⇒ LCS = 0

dp[N-1, M-1]

↳ Length of LCS of s1[0, N-1] & s2[0, M-1].

TC : O(NM)

SC : O(NM)

	0	1	2	3	4	5
S_1:	M	A	I	C	A	
S_2:	I	A	I	Y	A	S

N = 5, M = 6.

LCS: AIA ⇒ 3

dp[5][6]

		I	A	I	Y	A	S
		0	1	2	3	4	5
0	M	0	0	0	0	0	0
1	A	0	1	1	1	1	1
2	I	1	1	2	2	2	2
3	C	1	1	2	2	2	2
4	A	1	2	2	2	3	3

final ans.

dp[4,5]

S_1: 0 to N-1

S_2: 0 to M-1

ans: A I A

i-1, j-1	i-1, j
i, j-1	i, j

```
if (S1[i] == S2[j]) {
    dp[i,j] = 1 + dp[i-1, j-1];
}
else {
    dp[i,j] = max(dp[i-1, j], dp[i][j-1])
}
```

⇒ Print the LCS.

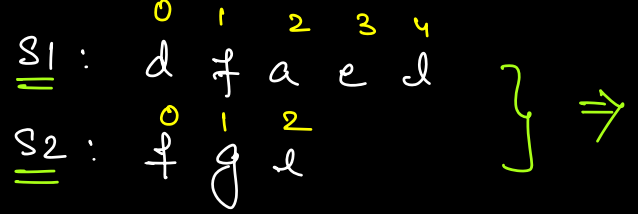
```
i = N-1, j = M-1;
String ans = "";
while (i >= 0 && j >= 0) {
    if (s1[i] == s2[j]) {
        ans += s1[i]
        i--, j--
    }
    else {
        if (i>0 && dp[i][j] = dp[i-1][j]) i--
        else j--
    }
}
return reverse(ans);
```

Q. Edit Distance

Given 2 string S1 & S2, Minimum no. of operations to be performed on S1 to make it S2.

In 1 operation on S1,

- We can insert 1 char at any position in S1.
- We can replace 1 char at any pos, with any char in S_1
- We can delete 1 char at any position in S1.



S_1 : d f ✗ ✗ e ⇒ (4)
↓ ↓
g g

S_1 : ✗ f a ✗ e ⇒ (3)
↓
g

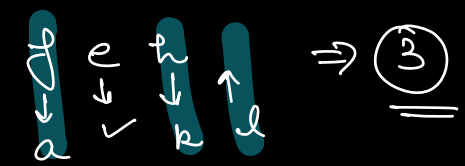
S_1 : ✗ f ✗ e e ⇒ (3)
↓
g

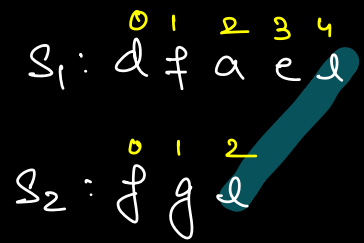
} = 3

Ex

S_1 : d e h

S_2 : a e k l





Ed (S_1[0-4], S_2[0-2])

if (S_1[4] == S_2[2]

↓

Ed (S_1[0-3], S_2[0-1])

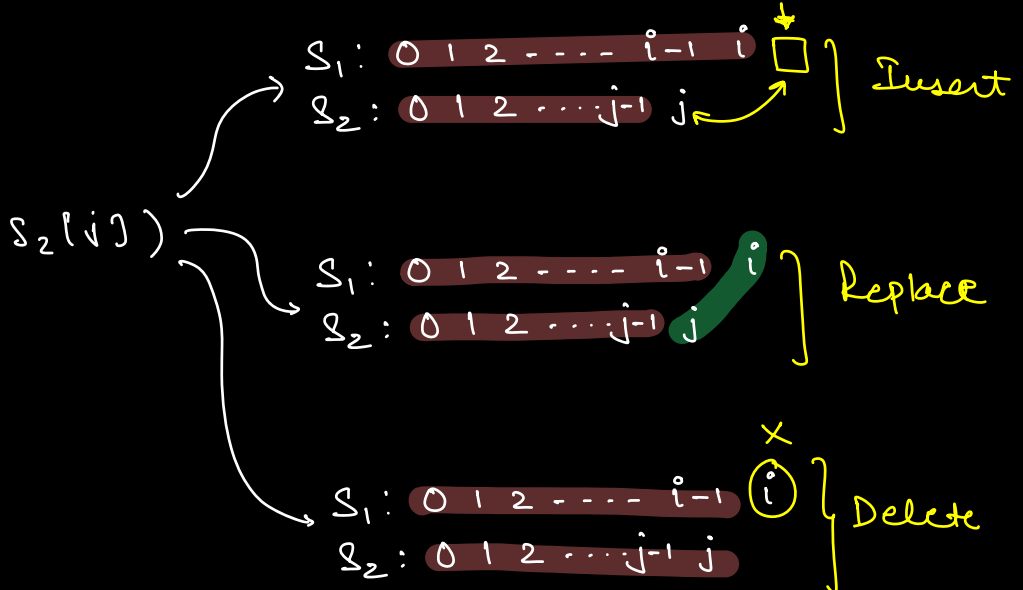
if (S_1[3] == S_2[1])

Insert (+1) → Ed (S_1[0-3], S_2[0-0])

Replace (+1) → Ed (S_1[0-2], S_2[0-0])

Delete (+1) → Ed (S_1[0-2], S_2[0-1])

if (S_1[i] != S_2[j])



dp[i,j] : Min # of operations required in S_1 to make to make $S_1 = S_2$

#

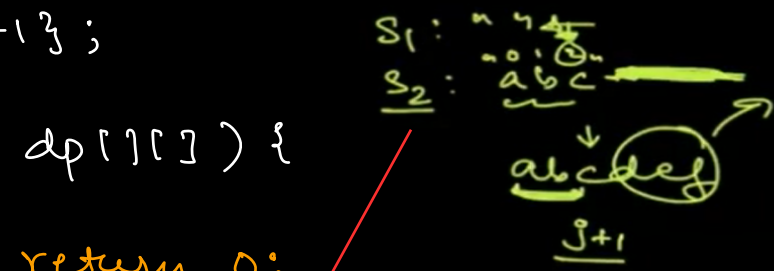
$$dp[i,j] = \begin{cases} \text{if } (S_1[i] == S_2[j]) \\ \quad dp[i,j] = dp[i-1][j-1] \\ \text{else} \{ \\ \quad dp[i][j] = 1 + \min \begin{cases} dp[i, j-1] \\ dp[i-1, j-1] \\ dp[i-1, j] \end{cases} \\ \} \end{cases}$$

S_1, S_2

int dp[N][M] = -1;

Ed(S_1, S_2, N-1, M-1, dp) ⇒ Call in Main()

int dp[N-1][M-1] = {-1};




```
int Ed(S1, S2, i, j, dp[][]) {
    (N)  (M)
    if (i == -1 && j == -1) return 0;
    if (i == -1) { return j+1; }   → insertions.
    if (j == -1) { return i+1; }   → deletions.

    if (dp[i][j] = -1) {
        if (S1[i] == S2[j]) {
            dp[i,j] = Ed(S1,S2, i-1, j-1, dp);
        }
        else {
            dp[i,j] = 1 + Min { Ed(S1,S2,i,j-1,dp)
                              { Ed(S1,S2,i-1,j,dp)
                              { Ed(S1,S2,i-1,j-1,dp)
        }
    }
    return dp[i,j];
}
```

TC: O(N x M)

SC: O(N x M)