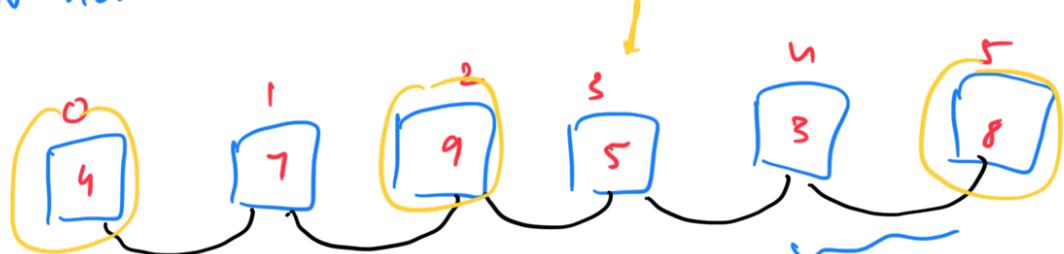


DP - 2

Question: Robbery

N houses which are connected by alarms



MaxGold \Rightarrow

$$7 + 9 + 8 = 24$$

maxGold(2) \Rightarrow maxGold(0)

maxGold(3) \Rightarrow maxGold(0, 1)

A =	0	3	4	2	1	5
	0	-	2	-	1	-

$$3 + 2 + 1 = 6$$

$$\Rightarrow 0 + 3 + 6 = 9$$

Assumption: 2
maxGold(i) \Rightarrow

Returning that we can maximum gold steal till i^{th} house

maxGold(5)

maxGold($n-1$)

gold > 0

A = 0 1 2
10 3 6 Ans =

0	1	2	3	4	5	6	7
10	3	6	2	1	5	8	9

maxGold(1)

0	1	2	3	4	5	6	7
10	3	6	2	1	5	8	9

maxGold(0)

maxGold(1) = $\max(A[0], A[1])$.

choice :

- 1) Steal from the i^{th} house
- gold = $A[i] + \text{maxGold}(i-1)$

, i^{th} house

dp[N]



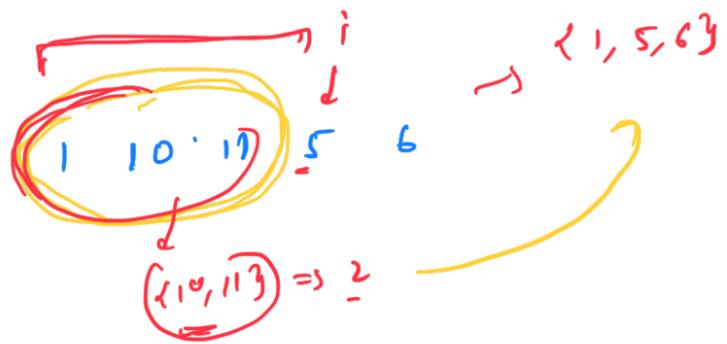
2) Don't steal from i
 $\text{gold}_i = \max_{\text{gold}}(i-1)$
 $\max_{\text{gold}}(i) = \max(\text{A}[i] + \max_{\text{gold}}(i-2),$
 $\max_{\text{gold}}(i-1))$

Overlapping Subproblems: YES
 Dynamic Programming

T.C: $O(\# \text{states} \times \text{T.C per state})$
 \downarrow
 $N \times O(1)$

T.C: $O(n)$
 S.C: $O(n) \rightarrow$
 \downarrow
 Top-Down
 \Rightarrow
 $O(1)$
 Bottom-Up

Question: Longest Increasing subsequence $\Rightarrow \text{Ans} = 4$
 $A = \begin{matrix} 0 & 1 & 2 & 3 \\ 1 & 3 & 2 & 3 \\ 2 & 1 & 3 & \end{matrix}$
 $\xrightarrow{\quad}$
 $1 \quad 3 \quad 5 \rightarrow 3$
 $1 \quad 3 \quad 3 \quad 5$
 $1 \quad 2 \quad 3 \quad 5 \rightarrow \boxed{4}$



Assumption:

LIS(i) \Rightarrow length of longest ending at index i

A =

endry at u r b
0 1 2 3 4 5 6
| 3 2 1 5 2 5

$$115(3) = \{1, 2, 3\} \Rightarrow 3$$

$$L(s(u)) = \{1\} \Rightarrow$$

$$L_{IS}(2) = \{1, 2\} \rightarrow 2$$

$$L(S \cup \{6\}) = d(1, 2, 3, 5) = 3 \text{ m}$$

$A =$

115 ()

Diagram illustrating the four cases for the first move:

- Case 1: $\{2, 1, 3\}$
- Case 2: $\frac{2}{1, 3}$
- Case 2 (underlined): $\frac{2}{1, 2, 3}$
- Case 3: $\{1, 2, 3\}$

CH 1-2

L1S(L1S(3)

$$\{1, 2, 3\} \Rightarrow \boxed{3}$$

A 2

2

27

3

4

6

| L1S =

$$\begin{array}{ccccccccc} & 1 & 2 & 2 & \textcolor{red}{(3)}^c & - & - & - \\ \textcolor{blue}{\cancel{x}} & \cancel{1} & 1,2 & 1,2 & \{1,3,5\} & 1,3 & \{1,2\} & 1,2,3 & \{1,2,3\} \end{array}$$



$$L(1, 2, f) \Rightarrow 3$$

$$L(1, 5) \Rightarrow 2$$

$$L(1, 2, 3, 5) \Rightarrow 3 + 1 = 4$$

$$L(1, 2, 5) \Rightarrow 2 + 1 = 3$$

$$L(1, 3, 5) \Rightarrow 2 + 1 = 3$$

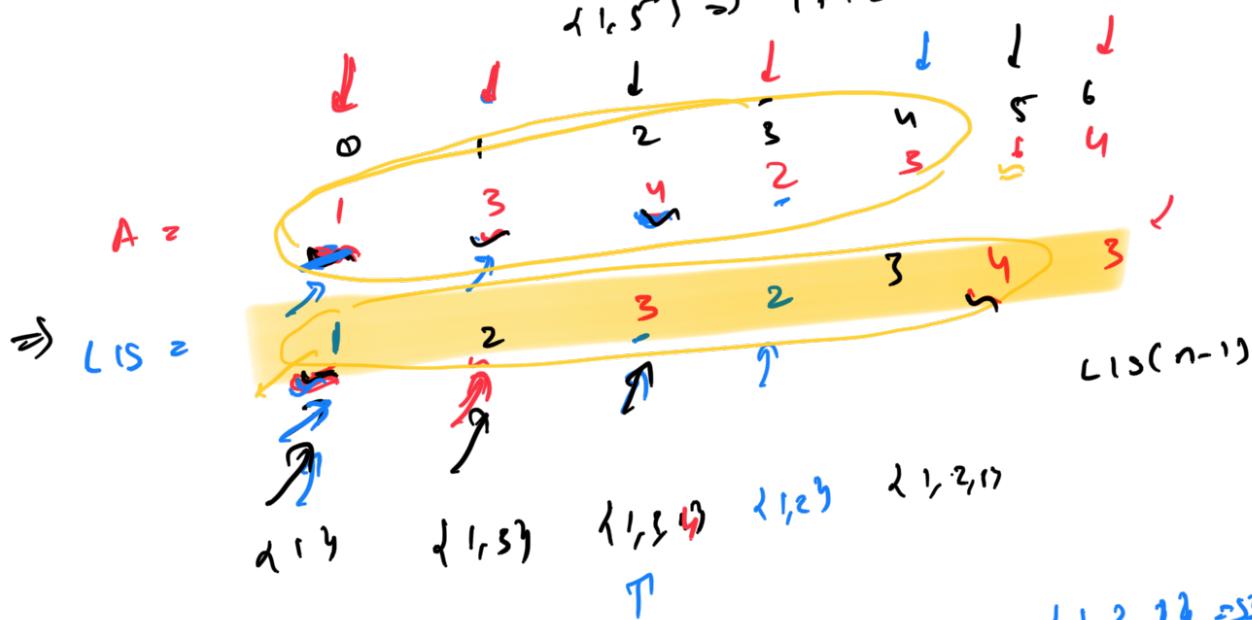
$$L(1, 5) \Rightarrow 1 + 1 = 2$$

$$\{1, 2\} \Rightarrow 2$$

$$\{1, 2\} \Rightarrow 2$$

$\rightarrow \boxed{\text{Max}}$

$$A[j] < \underline{A[i]}$$



$$\{1, 2, 3\} \Rightarrow 3$$

 $\}$
 $\{1, 3\} \Rightarrow 2$

Overlapping subproblems

$$LIS(s) \Rightarrow$$

$$LIS(4), LIS(3), LIS(2), LIS(1), LIS(0)$$

$$LIS(c) \Rightarrow s, t, \dots, 2$$

$$dp[j] = j-1$$

$$LIS(i)$$

Bottom-up

$$LIS[N] = \{ -1 \};$$

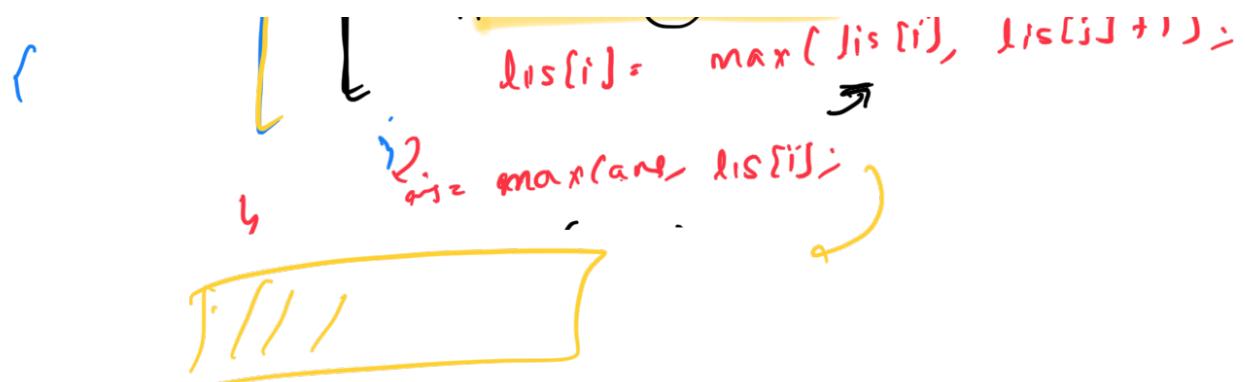
$$LIS[0] = 1$$

$$f \leq i-1 \Rightarrow j < i$$

```

for (i = 1; i < n; i++) { → LIS[i]
    LIS[i] = LIS[i-1];
    for (j = 0; j <= i-1; j++) {
        if (A[i] > A[j])
            ....
    }
}

```



$i =$
 $j = 0, \dots, i-1$
 $\max(\text{lis}[i], \dots)$
 $A[i] > A[j]$
 i
 $(0, \dots, i-1)$

~~Top-Down~~
 $O(n)$

```

LIS ( int i = 0 ) {
    if ( i == 0 ) return 1;
    ans = INT_MIN;
    for ( j = i - 1; j >= 0; j-- ) {
        if ( A[i] > A[j] ) {
            ans = max( ans, LIS(j) + 1 );
        }
    }
    precis = ...;
}

```

~~T.C:~~ $O(\# \text{states}) \times T.C \text{ per state}$
 n
 n

$T.C: O(n^2)$
 $S.C: O(n)$

Question: Print the LIS

$1, 2, 3, 4, 5, 6, ?$

$A =$

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

 $LIS = \{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$

S.C.: $O(n^2)$

$A =$

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

 $LIS = \{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$

$1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$

$A =$

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

 $LIS = \{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$

prev

LIS = $O(n) \approx n = 4$

$A =$

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

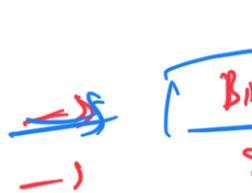
 $LIS = \{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$ $\{1, 2, 3, 4, 5\}$

prev

LIS = $O(n)$

S.C.: $O(n)$

Homework


Binary Search [Upper bound]
Segment Tree
}

Question: Longest Common Subsequence

$s_1 = \underline{a} b b c d f$

$s_2 = b \underline{a} b d c f$

$abcd, abdf, bbdf$

Ans = 4

$s_1 = a g g t a t$

$s_2 = g x t x a y b$

gtat = 3

$s_1 = \overbrace{a b c d}$
 $s_2 = \overbrace{d c b a}$

LCS(s₁, s₂) = 1

Recursion?

Assumption:

$LCS(l_1, l_2) =$ Length of longest common subseq
 of s_1 upto l_1 and s_2 upto l_2

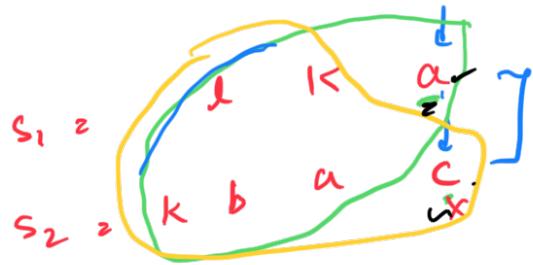
Case 1: Last characters of s_1 and s_2 are equal
 $s_1 = a b b c d f$
 $s_2 = b a b d c f$

$$l_1 = 6$$

$$l_2 = 6$$

$$LCS(l_1, l_2) = 1 + LCS(l_1-1, l_2-1);$$

Case 2: last characters are not equal



$$LCS(s_1, s_2) = k_a$$

$a \neq c$
 $a \neq c$

Option 1:

$$\begin{aligned} s_1 &= l \quad \underline{K} \quad a \\ s_2 &= \quad K \quad b \quad a \end{aligned}$$

$$LCS(l_1, l_2-1)$$

Option 2:

$$\begin{aligned} s_1 &= \quad l \quad \underline{K} \quad LCS(l_1-1, l_2) \\ s_2 &= \quad K \quad b \quad a \quad c \end{aligned}$$

$$LCS(l_1, l_2)$$

if ($s_1[l_1-1] == s_2[l_2-1]$) {

$$LCS(l_1, l_2) = 1 + LCS(l_1-1, l_2-1);$$

}

else {

$$LCS(l_1, l_2) = \max(LCS(l_1-1, l_2),$$

$$LCS(l_1, l_2-1))$$

top-down

$$dp[l_1] \cap l_2] = \infty$$

if ($l_1 = 0$)

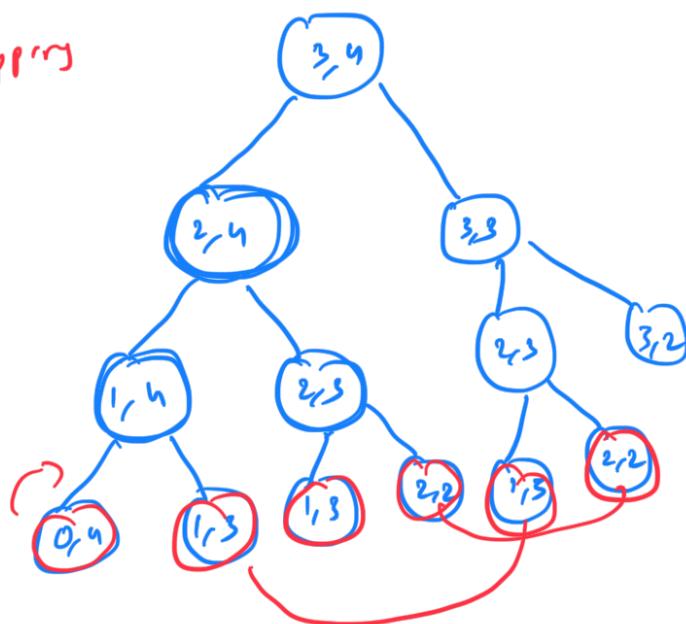
$$LCS(l_1, l_2) = 0$$

if ($l_2 = 0$)

$$LCS(l_1, l_2) = 0$$

$S_1 = "r"$
 $S_2 = "abcd"$

Overlap



$S_1 = ^{0 \ 1 \ 2} k \ K a$
 $S_2 = ^{0 \ 1 \ 2 \ 3} k b a c$

[dp] → [] 1D array

[TOP-DOWN] →

Bottom - UP

Σ
 $2D = \square$
 $dp[\Sigma_i][j]$

<img alt="A 5x5 grid representing a 2D DP state. The columns are labeled 0, 1, 2, 3, 4 and the rows are labeled 0, 1, 2, 3, 4. The grid contains various symbols: '0' at (0,0), (1,0), (2,0), (3,0); '1' at (0,1), (1,1), (2,1), (3,1), (4,1); '2' at (0,2), (1,2), (2,2), (3,2); '3' at (0,3), (1,3), (2,3), (3,3); '4' at (0,4), (1,4), (2,4), (3,4); 'S1' at (0,0), (1,0), (2,0); 'S2' at (0,1), (1,1), (2,1); 'K' at (0,2), (1,2), (2,2); 'a' at (0,3), (1,3), (2,3), (3,3); 'b' at (0,4), (1,4), (2,4). Red annotations include arrows pointing from '0' to '1', '1' to '2', '2' to '3', and '3' to '4'. A yellow shaded row covers the bottom row (4,0) to (4,4). A blue arrow points from the bottom right corner (4,4) to the formula <math>dp[l_1][l_2]. Handwritten formulas on the right side include:
 $dp[2][3][4] \quad s_1 = lka \quad l_1 = 3$
 $s_2 = kbac \quad l_2 = 4$
 $l_1 = 0 \quad l_2 = 0 \quad dp[l_1][l_2] = 0$
 $dp[1][2][3]$
 $\max(dp[0, -1][l_2], dp[1, -1][l_1, -1])$
 $1 + dp[l_1, -1][l_2, -1]$
 $s_1 = lka$
 $s_2 = kbac$
 $dp[3][4] \text{ or } dp[2][4]$

$dp[i_1+1][i_2+1];$
 for ($i \geq 0; i < l_1; i++$)
 $dp[i][0] = 0;$
 for ($i \geq 0; i < l_2; i++$)
 $dp[0][i] = \infty$
 for ($i \geq 1; i \leq l_1; i++$) {
 for ($j \geq 1; j \leq l_2; j++$) {
 if $s_1[i-1] == s_2[j-1]$
 $dp[i][j] = 1 + dp[i-1][j-1]$
 else
 $dp[i][j] = \min(dp[i-1][j-1],$
 $dp[i-1][j])$

T.C: $O(\# \text{state} \times \text{T.C per state})$
 \downarrow
 $l_1 \times l_2$
 $O(\min(l_1, l_2))$
 S.C: $O(l_1 \times l_2) \rightarrow O(l_1)$
 Bottom Up

Question: Edit Distance

$s_1 = \text{sunday} \rightarrow)$
 $s_2 = \text{saturday}$

Convert s_1 to s_2 using minimum no. of
 operations

- 1) Insert in S_1
- 2) Replace a char in S_1
- 3) Delete a character in S_1

$S_1 = \begin{matrix} \text{S} \\ \text{a} \\ \text{t} \\ \text{u} \\ \text{r} \\ \text{d} \\ \text{y} \end{matrix}$ day \rightarrow Saturday

$S_2 = \begin{matrix} \text{s} \\ \text{a} \\ \text{t} \\ \text{u} \\ \text{r} \\ \text{d} \\ \text{y} \end{matrix}$ day

strategy 1:

rep(u, t) $\rightarrow R_u(n, t)$:- insert(u), insert(t)
 n operators

strategy 2: insert(a), insert(t), rep(n, r)
 3 operators

$S_1 = \begin{matrix} \text{T} \\ \text{u} \\ \text{e} \\ \text{s} \\ \text{d} \\ \text{y} \end{matrix}$ day } 2 operators
 $S_2 = \begin{matrix} \text{T} \\ \text{h} \\ \text{u} \\ \text{r} \\ \text{s} \\ \text{d} \\ \text{y} \end{matrix}$ day }

$S_1 = \begin{matrix} \text{c} \\ \text{d} \\ \text{a} \\ \text{b} \end{matrix}$ }]
 $S_2 = \begin{matrix} \text{c} \\ \text{d} \\ \text{a} \end{matrix}$ ← $\boxed{3 - 1 = 2}$
 3 operators $\boxed{\text{len}(S_2) - \text{lcs}(S_1, S_2)}$

$\min \text{lost}(l_1, l_2) \Rightarrow$ Gives the minimum no of operations upto l_1 and S_2
 To convert S_1 upto l_2

Case 1: Last characters are equal
 round a(y)

$$S_1 = \{ \dots \}$$

$$S_2 = \{ \text{saturday} \}$$

$$\min \text{cost}(l_1, l_2) = \min \text{cost}(l_1 - 1, l_2 - 1);$$

Case 1: Last characters are not equal

$$S_1 = \begin{matrix} \text{s} & \text{u} & \text{n} \\ \cancel{\text{a}} & \cancel{\text{f}} & \text{u} \end{matrix}$$

$$S_2 = \begin{matrix} \text{s} & \text{a} & \text{f} & \text{u} & \cancel{\text{y}} \end{matrix}$$

Option 1: Insert

$$\begin{matrix} \text{s} & \text{u} & \text{n} & \cancel{\text{y}} \\ \text{s} & \text{a} & \text{f} & \text{u} & \cancel{\text{y}} \end{matrix}$$

$$1 + \min \text{cost}(l_1, l_2 - 1)$$

Option 2: Replace

$$S_1 = \begin{matrix} \text{s} & \text{u} & \cancel{\text{n}} & \cancel{\text{y}} \\ \text{s} & \text{a} & \text{f} & \text{u} \end{matrix}$$

$$S_2 = \begin{matrix} \text{s} & \text{a} & \text{f} & \text{u} & \cancel{\text{y}} \end{matrix}$$

$$1 + \min \text{cost}(l_1 - 1, l_2 - 1)$$

Option 3: No delete

$$S_1 = \begin{matrix} \text{s} & \text{u} & \cancel{\text{x}} \\ \text{s} & \text{a} & \text{f} & \text{u} & \cancel{\text{y}} \end{matrix}$$

$$1 + \min \text{cost}(l_1 - 1, l_2);$$

```

int minCost(l1, l2) {
    if (l1 == 0) return l2;
    if (l2 == 0) return l1;
    if (dp[l1][l2] != -1) return dp[l1][l2];
    if (S1[l1 - 1] == S2[l2 - 1]) {
        noCost[l1][l2] = minCost(l1 - 1, l2 - 1);
    } else {
        noCost[l1][l2] = 1 + minCost(l1, l2 - 1);
    }
}

```

```

    }  

    else {  

        dp(l1, l2) = 1 + min ( minCost(l1, -1, l2)  

                                minCost(l1, -1, l2-1)  

                                minCost(l1, -1, l2-2))  

        return dp(l1, l2)
    }
}

```

$$\begin{array}{l} \Sigma_1 = \{ \ldots \} \\ \Sigma_2 = \{ \text{a b c d e} \} \end{array} \quad \gamma \rightarrow \lambda_2$$

$$S_1 = \begin{matrix} "abcd" \\ a \\ 4 \end{matrix} \quad S_2 =$$

$$T.C.: \frac{O(\text{# states})}{l_1 \times l_2} \times O(1) \times T.C \text{ per stack}$$

$$\begin{array}{ll} T.C: & O(l_1 \cdot r l_2) \\ S.C: & O(l_1 \cdot r l_2) \end{array}$$

Bottom-up

$d_p[i+1][j_2+1]$

		S ₂								
		0	1	2	3	4	5	6	7	8
S ₁		0	1	2	3	4	5	6	7	8
0	,	0	1	2	3	4	5	6	7	8
1	s	1	0 ← 1	2	3	4	5	6	7	
2	u	2	1	1	2	2				
3	n	3								
4	d	4								
5	a	5								
6	y	6								

$i \leftarrow l_1 = 0$
 $ans = l_2$

$i \leftarrow l_2 = -1$
 $ans = l_1$

$l + \min$
 $d_p[l_1 - 1][l_2]$
 $d_p[l_1 - 1][l_2 - 1]$
 $d_p[l_1][l_2 - 1]$

// fill 1st row & 1st column

for ($i=1$; $i \leq l_1$; $i++$) {

 for ($j=1$; $j \leq l_2$; $j++$) {

 if ($s_1[i-1] = s_2[j-1]$) {

$dp[i][j] = dp[i-1][j-1];$

 }

 else

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

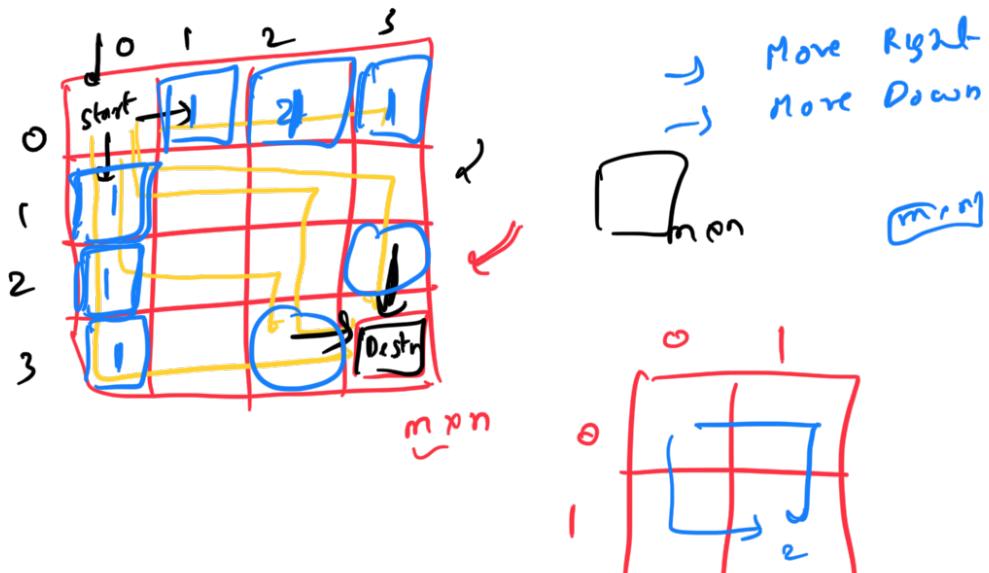
$dp[i-1][j-1]);$

}

7

$$T.C: O(l_1 \times l_2)$$
$$S.C: O(l_1 + l_2) \rightarrow O(l_1)$$

Question: ways to reach



Approach: Stairs

$\text{ways}(i, j) \Rightarrow$ No. of ways to reach the cell (i, j) from $(0, 0)$

i.e., $\text{ways}(i, j) = \text{ways}(i-1, j) + \text{ways}(i, j-1)$

$$\{ \text{ways}(i, j) = \text{ways}(i-1, j)$$

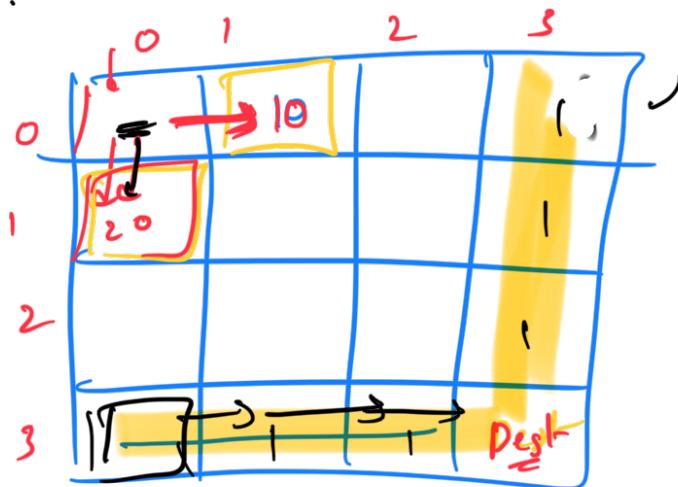
$i = 0 \text{ or } j = 0$
return 1

$$T.C: O(mn \times O(1)) = O(mn)$$

$$S.C: O(mn) \rightarrow O(m)$$

\downarrow
Bottom-up

Approach 2 :



$\text{ways}(i, j) = \text{No. of ways to reach destination}$
 $(n-1, m-1)$ from (i, j)

$$\text{ways}(i, j) = \text{ways}(i+1, j) + \text{ways}(i, j+1)$$

Base Case

$$(i = n-1 \text{ or } j = m-1) \\ \text{ways}(i, j) = 1$$

Question : Robbery

```
dp[N] = {-1};  
int maxGold(int gold[], int N){  
    if(dp[n] != -1) return dp[n];  
    if(n == 0) return dp[n] = A[0];  
    if(n == 1) return dp[n] = max(A[0], A[1]);  
  
    dp[n] = max(gold[N] + maxGold(N-2), maxGold(N-1));  
    return dp[n];  
}  
ans = maxGold(gold, N-1); // N = gold.length();
```

Longest Increasing Subsequence

```
dp[0] = 1; // Smaller Problem  
for(int i = 1; i < n; i++){  
    dp[i] = 1;  
    for(int j = i-1; j >= 0; j--){  
        if(A[j] < A[i]){  
            dp[i] = max(dp[i], dp[j] + 1);  
        }  
    }  
}
```

Longest Common Subsequence: Top Down

```
dp[la+1][lb+1] = {-1};  
int lcs(string A, string B, int la, int lb){  
    if(la == 0 || lb == 0) return 0;  
  
    if(dp[la][lb] != -1) return dp[la][lb];  
  
    if(A[la - 1] == B[lb - 1])  
        dp[la][lb] = 1 + lcs(A, B, la-1, lb-1);  
    else  
        dp[la][lb] = max(lcs(A, B, la-1, lb), lcs(A, B, la, lb-1));  
    return dp[la][lb];  
}
```

Longest Common Subsequence: Bottom-Up

```
int LCS(string A, string B){  
    la = A.length();  
    lb = B.length();  
    int lcs[la+1][lb+1];  
    for(int i = 0; i < la; i++) lcs[i][0] = 0;  
    for(int i = 0; i < lb; i++) lcs[0][i] = 0;  
  
    for(int i = 1; i <= la; i++){  
        for(int j = 1; j <= lb; j++){  
            if(A[i-1] == B[j-1])  
                lcs[i][j] = lcs[i-1][j-1] + 1;  
            else  
                lcs[i][j] = max(lcs[i-1][j], lcs[i][j-1]);  
        }  
    }  
    return lcs[la][lb];  
}
```

Edit Distance : Top Down

```
edit(A, B, 11, 12){  
    if(11 == 0) return 12;  
    if(12 == 0) return 11;  
  
    if(dp[11][12] != -1) return dp[11][12];  
    if(A[11 - 1] == B[12 - 1])  
        dp[11][12] = edit(A, B, 11-1, 12 - 1);  
    else{  
        dp[11][12] = 1 + min( edit(A, B, 11, 12 - 1),  
                               edit(A, B, 11-1, 12 - 1),  
                               edit(A, B, 11-1, 12))  
    }  
    return dp[11][12];  
}
```

Top-Down : Ways to reach (r-1, c-1) from (0, 0)

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

Bottom-up: Ways to reach (r-1, c-1) from (0, 0)

```
int ways(int r, int c){  
    ways[r][c] = {0};  
    ways[0][0] = 1;  
    for(int i = 0; i < r; i++) ways[i][0] = 1;  
    for(int i = 0; i < c; i++) ways[0][i] = 1;  
  
    for(int i = 1; i < r; i++){  
        for(int j = 1; j < c; j++){  
            ways[i][j] = ways[i-1][j] + ways[i][j-1];  
        }  
    }  
    return ways[r-1][c-1];  
}
```