j \ i	1	2	3	4
1	0	1750 k=1	4375 K=2	7000 K=3
2		0	3750 K=1	7500 K=3
3			0	9375 K=1
4				0

A	B	C	D
7x10	10x25	25x15	15x25
$d_0\ d_1$	$d_2\ d_3$	$d_4\ d_5$	$d_6\ d_7$

$$C_{1,2} = C_{1,1} + C_{2,2} + d_0 d_1 d_2$$

$$= 0 + 0 + 1750 = 1750$$

$$C_{2,3} = 0 + 0 + 3750 = 3750$$

$$C_{3,4} = 0 + 0 + 9375 = 9375$$

$$C_{1,3} = \min\{C_{1,k} + C_{k+1,3} + d_0 d_k d_3\}$$

$$= \min \begin{cases} k=1 \longrightarrow 0 + 3750 + \overset{1050}{\cancel{7\times10\times15}} = 4800 \\ k=2 \longrightarrow 1750 + 0 + \overset{2625}{\cancel{7\times25\times15}} = 4375 \end{cases}$$

$$C_{2,4} = \min\{C_{2,k} + C_{k+1,4} + d_1 d_k d_4\}$$

$2 \leq k < 4$

$$= \min \begin{cases} k=2 \longrightarrow 0 + 9375 + \overset{6250}{\cancel{10\times25\times25}} = 15625 \\ k=3 \longrightarrow 3750 + 0 + \overset{3750}{\cancel{10\times15\times25}} = 7500 \end{cases}$$

$$C_{1,4} = \min\{C_{1,k} + C_{k+1,4} + d_0 d_k d_4\}$$

$1 \leq k < 4$

$$= \min \begin{cases} k=1 \longrightarrow 0 + 7500 + \overset{1750}{\cancel{7\times10\times25}} = 9250 \\ k=2 \longrightarrow 1750 + 9375 + \overset{4375}{\cancel{7\times25\times25}} = 15500 \\ k=3 \longrightarrow 4375 + 0 + \overset{2625}{\cancel{7\times15\times25}} = 7000 \end{cases}$$

الف)

a	b		1	2	2	3	4	4	5	5	5
		0	1	2	3	4	5	6	7	8	9
	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	1	1	1	1	1	1
1	2	0	1	1	1	1	1	1	1	1	1
4	3	0	1	1	1	1	2	2	2	2	2
2	4	0	1	2	2	2	2	2	2	2	2
2	5	0	1	2	3	3	3	3	3	3	3
4	6	0	1	2	3	3	4	4	4	4	4
3	7	0	1	2	3	4	4	4	4	4	4
4	8	0	1	2	3	4	5	5	5	5	5
5	9	0	1	2	3	4	5	5	6	6	6

LCS ⟶ [1, 2, 2, 3, 4, 5]

ب) همانطور که در ساختار بالا می‌بینید تعداد زیر مسئله $(n+1)(m+1)$ چون فرض می‌کنیم 0 خانه الی n خانه از آرایه اول و 0 خانه الی m خانه از آرایه دوم را داریم و برای تمامی آنها مسئله را حل می‌کنیم.

تعداد کل زیر مسائل $= (n+1)(m+1) = O(m \cdot n)$

Scanned with CamScanner

```
LPath(V, E; S, D)
    dist = [ -MAX_INT] * len(V)          // negative infinite
    dist[S] = 0   // Source = 0

    G = {V, E}   // create graph
    topologicalOrder(G)

    for Vex in V do
        for u in E do
            if dist[Vex] < dist[u] + weight(U, Vex) do
                dist[Vex] = dist[u] + weight(U, Vex)
            end if
        end for
    end for
    print(dist)
```

چون در dist تمای مقادیر گره های قبلی، مقدار دهی شده اند dp است.

Time complexity: $O(V+E)$ → order این که تو در تو هستند داریم for ۲تا است.

(4) ۲ راه وجود دارد.

راه اول: فقط از insert و delete استفاده کنیم.

اگر فرض کنیم تابع LCS داریم که بزرگ‌ترین زیر دنباله مشترک را حساب می‌کند.

```
dp[m][n] = -1
LCS(S1, S2, m, n):      // m = len(S1), n = len(S2)
    if i == 0 || j == 0 do
        return 0
    if dp[i][j] != -1 do
        return dp[i][j]
    if S1[i-1] == S2[j-1] do
        return dp[i][j] = 1 + LCS(S1, S2, i-1, j-1)
    else
        return dp[i][j] = max(LCS(S1, S2, i, j-1),
                              LCS(S1, S2, i-1, j))
```

Time complexity = $O(mn)$

الگوریتم راه ۱:

```
Convert(S1, S2)
    m = len(S1)
    n = len(S2)
    dp[m][n] = -1
    SC = LCS(S1, S2, m, n)

    LL = len(SC)
    return m - LL + n - LL
```

هزینه را برای حالتی که فقط اضافه و کم می‌کند را خروجی می‌دهد.

Scanned with CamScanner

راه دوم : فقط از Replace و Delete, استفاده کنیم. مثلاً
Insert

$S1 = a\,a\,b\,c\,a$

$S2 = a\,b\,b\,c\,d\,e\,f$ → insert

Replace

Replace a with b

```
Convert (S1, S2)
    m = len(S1)
    n = len(S2)
    dp[m][n] = -1
    SC = LCS (S1,
    for i in range o, .. do
        if S1[i] != S2[i] do
            S1[i] = S2[i]          // Replace
    if m < n do
        for i in range m+1, n do
            S1[i] = S2[i]          // insert
    else
        for i in
```

هزینه کل = تفاوت . min L + o, هزینه Delete / Insert

پس در بدترین حالت :

$$O(\min(m,n)) + O(n-m) = O(n)$$

1) Generate all 6 rotations of all boxes. The size of the rotation array becomes 6 times the size of the origin array.

2) Sort all the above generated 6n boxes in decreasing order

3) After Sorting the boxes, the problem is Same as LIS with following optimal Substructure property.

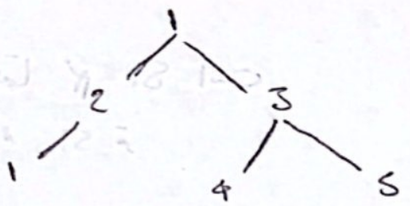
MSH(i) = Maximum possible stack height with box i at top of stack

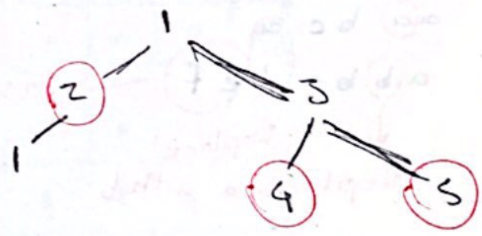
MSH(i) = { Max(MSH(j)) + height(i) }

where j<i and sideup(j) = sidedown(i)

4) To get overall max height, we return max(MSH(i)) where 0<i<n

(b) فرض می‌کنیم Binary Tree داریم و نباید همسایه‌های مستقیم را انتخاب کنیم.





از یک dict به عنوان dynamic Array استفاده می‌کنیم تا اطلاعات را در آن ذخیره کنیم، و از آنها استفاده، update کنیم. dynamic programming است.

```
map = {}
maxSum(root):
    if root = NULL do
        return 0
    if root in map do
        return map[root]
    x = root.data
    if root.left != NULL
        x += maxSum(root.Left.Left) + maxSum(root.left.right)
    if root.right != NULL
        x += maxSum(root.right.left) + maxSum(root.right.left)
    curr = maxSum(root.left) + maxSum(root.right)
    map[root] = max(x, curr)
    return max(x, curr)
```

Time Complexity O(n)

Space " O(n)