Dynamic Programming:

↳ Dynamic programming, like the divide-and-conquer method, solves problems by combining the solutions to sub problems.

↳ Divide-and-conquer algorithms partition the problem into independent sub problems, solve the sub problems recursively, and then combine their solutions to solve the original problem.

↳ In contrast, dynamic programming is applicable ~~with~~ when the sub problems are not independent, that is, when sub problems share sub problems.

↳ In this context, a divide-and-conquer algorithm does ~~not~~ more work than necessary, repeatedly solving the common sub problems.

↳ A dynamic-programming algorithm solves every sub problem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time the sub problem is encountered.

↳ The development of a dynamic-programming algorithm can be broken into a sequence of four steps.

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution in a bottom-up fashion.
4. Construct an optimal solution from computed information.

Principle of optimality:

↳ The dynamic programming algorithm obtains the solution using principle of optimality.

↳ The principle of optimality states that "in an optimal sequence of decision or choices, each subsequence must also be optimal."

↳ When it is not possible to apply the principle of optimality it is almost impossible to obtain the solution using the dynamic programming approach.

↳ The principale of optimality:- "If k is a node on the shortest path from i to j, then the part of the path from i to k, and the part from k to j, must also be optimal."

Calculating the Binomial coefficient:

↳ Calculating Binomial co-efficients can be important for solving combinatorial problems. A formula for computing binomial co-efficients is this:

$$\binom{n}{m} = \frac{n!}{(n-m)!\,m!}$$

↳ using an identity called Pascal's Formula a recursive formulation for it looks like this:

$$\binom{n}{m} = \begin{cases} 1 & \text{if } m = 0 \\ 1 & \text{if } n = m \\ \binom{n-1}{m} + \binom{n-1}{m-1} & \text{otherwise} \end{cases}$$

↳ This constructions forms

n	$\binom{n}{0}$	$\binom{n}{1}$	$\binom{n}{2}$	$\binom{n}{3}$	$\binom{n}{4}$	$\binom{n}{5}$	$\binom{n}{6}$	$\binom{n}{7}$
0	1							
1	1	1						
2	1	2	1					
3	1	3	3	1				
4	1	4	6	4	1			
5	1	5	10	10	5	1		
6	1	6	15	20	15	6	1	
7	1	7	21	35	35	21	7	1

↳ Each number in the triangle is the sum of the two numbers directly above it.

★ Making change problem using dynamic programming

★ Algorithm:

```
function coins (N)
{ Gives the minimum number of coins needed to make
  change for N units.
  Array d[1...n] specifies the coinage: in the example
  there are coins for 1, 4 and 6 units. }

  array d[1...n] = [1, 4, 6]
  array c[1...n, 0...N]
  for i ← 1 to n do c[i, 0] ← 0
  for j ← 1 to n do
      for j ← 1 to N do
          c[i, j] ← if i = 1 and j < d[i] then +∞
                    else if i = 1 then 1 + c[1, j - d[1]]
                    else if j < d[i] then c[i-1, j]
                    else min (c[i-1, j], 1 + c[1, j-d[i]])
  return c[n, N]
```

→ We need to generate table c[n][N], where,

n = number of denominations.

Here we are having 3 denomination so n = 3.

N = number of units that you need to make change

Here we need change of 8 units so N = 8

↳ To generate table c[i][j] use following steps

Step-1 : Make c[i][0] = 0 for $0 < i \leq n$

Step-2 : Repeat Step-2 to Step-4 for remaining matrix values
if i=1 then c[i][j] = 1 + c[1][j-d1], here $d_1 = 1$

Step-3 : if j < di then c[i][j] = c[i-1][j]

Step-4 : otherwise c[i][j] = min(c[i-1][j], 1+c[i][j-di])

* Example: Denomination $d_1 = 2$, $d_2 = 4$, $d_3 = 6$ Make change of Rs. 8.

Soln:

c[i][j] = c[3][8]

	j → / i ↓	0	1	2	3	4	5	6	7	8
$d_1 = 1$	1	0	1	2	3	4	5	6	7	8
$d_2 = 4$	2	0	1	2	3	1	2	3	4	2
$d_3 = 6$	3	0	1	2	3	1	2	1	2	2

We need minimum c[3][8] = 2 coins for change

Step 2 : i = 1, j = 1, $d_1 = 1$

c[1][1] = 1 + c[1][1-1]
= 1 + c[1][0]
= 1

Step 3 : i = 1, j = 2, $d_1 = 1$

c[1][2] = 1 + c[1][2-1]
= 1 + c[1][1]
= 2

Similar, for i = 1 & j = 3 to j = 8 & $d_1 = 1$.

Steps - II 1. $j = 2$, $i = 1$, $d_2 = 4$

$\therefore i < d_2$ $(1 < 4)$

$\therefore c[i][j] = c[i-1][j]$

$c[2][1] = c[2-1][1]$

$= c[1][1]$

$= 1$

2. $j = 2$, $i = 2$, $d_2 = 4$

$i < d_2$ $(2 < 4)$

$c[i][j] = c[i-1][j]$

$= c[2-1][2]$

$= c[1][2]$

$= 2$

3. $j = 2$ $i = 3$, $d_2 = 4$

$\therefore i < d_2$ $(3 < 4)$

$c[2][3] = c[2-1][3]$

$= 3.$

4. $j = 2$, $i = 4$, $d_2 = 4$

$\therefore c[i][j] = \min(c[i-1][j], 1 + c[i][j-d])$

$c[2][4] = \min(c[1,4], 1 + c[2,4-4])$

$= \min(4, 1+0)$

$= 1$

5. $c[2][5] = \min(c[1,5], 1 + c[2,5-4])$

$= \min(5, 1+1)$

$= 2.$

6. $c[2][6] =$

$= \min(c[1,6], 1 + c[2,6-4])$

$= \min(6, 1+2)$

$= 3$

7. $c[2][7] = \min(c[1,7], 1 + c[2,7-4])$

$= \min(7, 1+3)$

$= 4$

8. $c[2][8] =$

$= \min(c[1,8], 1 + c[2,8-4])$

$= \min(8, 1+1)$

$= 2.$

Step - III 1. $j = 3$, $i = 1$, $d_3 = 6$

$i < d_3$ $(1 < 6)$

$c[i][j] = c[i-1][j]$

$= c[3-1][1]$

$c[3][1] = 1$

2. $c[3][2] = c[3-1][2]$

$= 2.$

Similarly, $j = 3$, $i = 3$ to $i = 8$ $d_3 = 6$

* Assembly line scheduling using dynamic programming

↳ Each line has n stations: $S_{1,1}, S_{1,2}, \ldots, S_{1,n}$ & $S_{2,1}, S_{2,2}, \ldots, S_{2,n}$

↳ Corresponding station $S_{1,j}$ & $S_{2,j}$ perform the same function but can take different amounts of time $a_{1,j}$ & $a_{2,j}$

↳ Entry times are: e_1 & e_2; exit times are: x_1 & x_2.

↳ After going through a station, can either:

- Stay on same line at no cost, or
- Transfer to other line: cost after $S_{j,j}$ is $t_{j,1}$ $j = 1, \ldots n-1$

Step-1. f^* : The structure of the fastest way through the factory.

$f_i[j]$: The structure of the fastest way through time to get from the starting point through station $S_{j,j}$

Step-2. A Recursive Solution:

$$f^* = \min (f_1[n] + x_1, f_2[n] + x_2)$$

Best case: $j = 1$, $j = 1, 2$ (getting through station 1)

$$f_1[1] = e_1 + a_{1,1}$$

$$f_2[1] = e_2 + a_{2,1}$$

General case: $j = 2, 3, \ldots, n$ & $j = 1, 2$

Fastest way through $S_{j,j}$ is either:

- The way through $S_{1,j-1}$ then directly through $S_{1,j}$.
  or $f_1[j-1] + a_{1,j}$

- The way through $S_{2,j-1}$, transfer from line 2 to line 1, then through $S_{1,j}$

$$f_2[j-1] + t_{2,j-1} + a_{1,j}$$

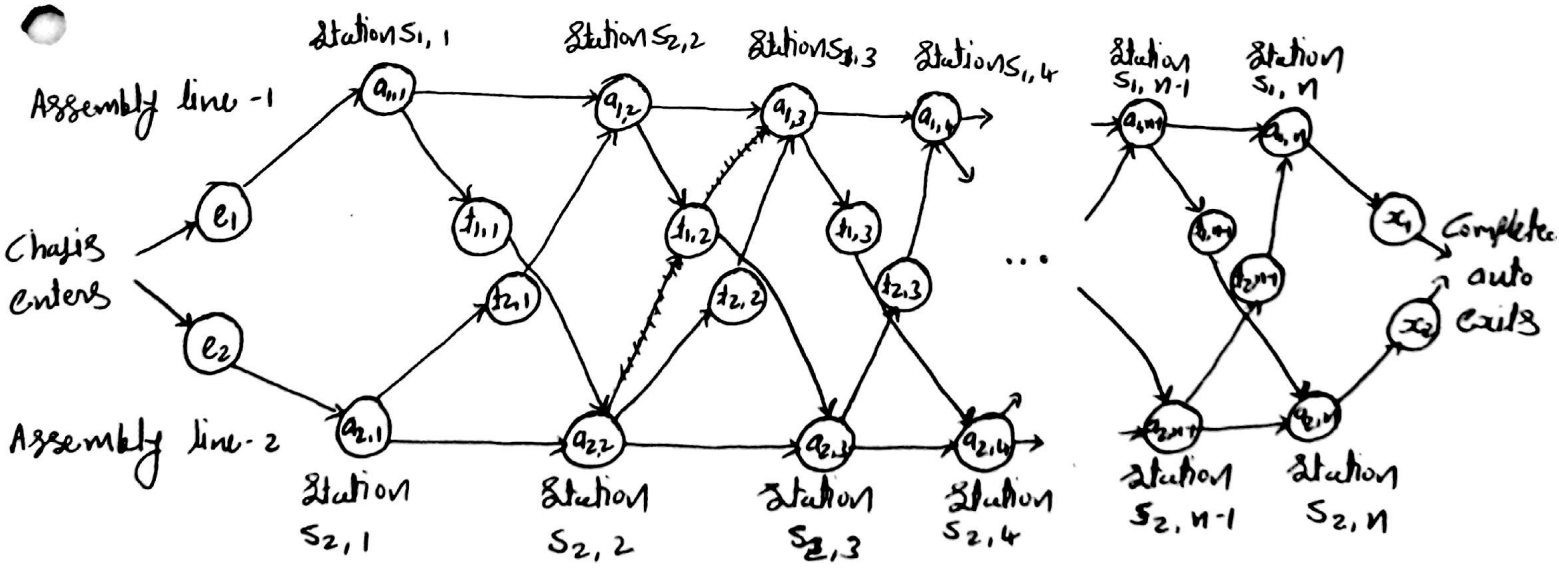
$$f_1[j] = \begin{cases} e_1 + a_{1,1}, & \text{if } j=1 \\ \min(f_1[j-1] + a_{1,j},\ f_2[j-1] + t_{2,j-1} + a_{1,j}) & \text{if } j \geq 2 \end{cases}$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1}, & \text{if } j=1 \\ \min(f_2[j-1] + a_{2,j},\ f_1[j-1] + t_{1,j-1} + a_{2,j}), & \text{if } j \geq 2 \end{cases}$$

Step - 3. Computing the fastest times

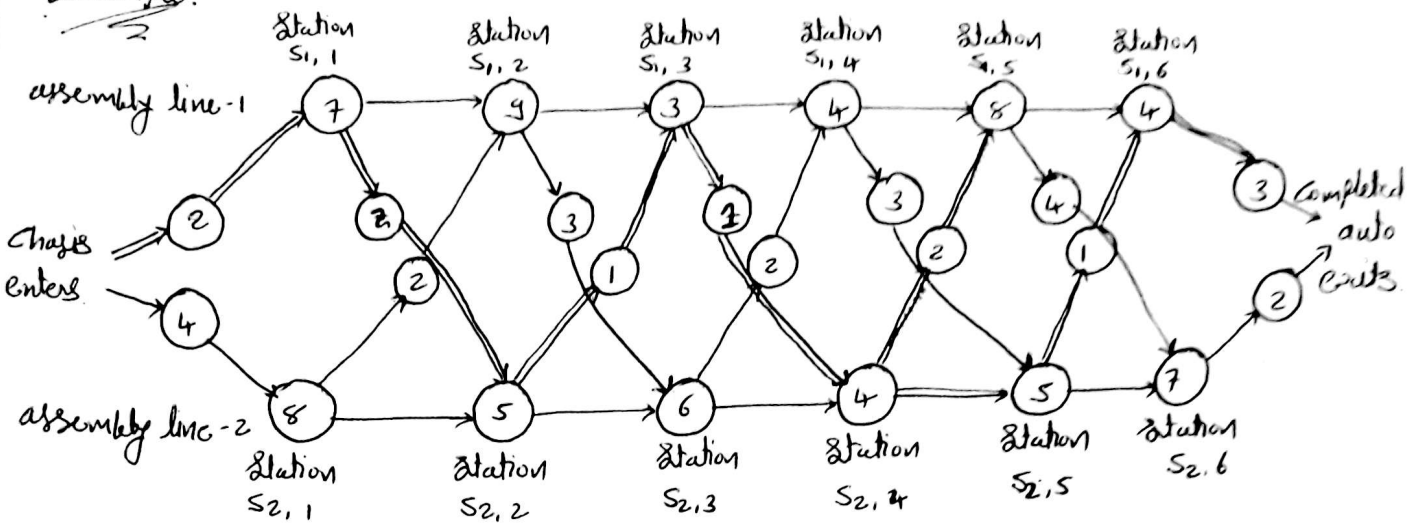
$$f^* = \min(f_1[n] + x_1,\ f_2[n] + x_2)$$

Step - 4. Constructing the fastest way through the factory.



Figure

Example:



$\div f_1[1] = e_1 + a_{1,1}, \; i=1$ $\quad \div f_2[1] = e_2 + a_{2,1}, \; i=1$

$f_1[1] = 2+7$ $\quad f_2[1] = 4+8 = 12.$

$= 9$

$\div f_1[2] = \min(f_1[i-1] + a_{1,i}, f_2[i-1] + t_{2,i-1} + a_{1,i}), \; i=2$

$= \min(9+9, 12+2+9)$

$= 18.$

$\div f_2[2] = \min(f_2[i-1] + a_{2,i}, f_1[i-1] + t_{1,i-1} + a_{2,i})$

$= \min(12+5, 9+2+5)$

$= 16.$

Repeating same formula for for

$f_1[3] \ldots f_1[6]$ & $f_2[3] \ldots f_2[6]$

$\hookrightarrow f^* = \min(f_1[n] + x_1, f_2[n] + x_2)$

$= \min(35+3, 37+2)$

$= 38$

i	1	2	3	4	5	6
$f_1[i]$	9	18	20	24	32	35
$f_2[i]$	12	16	22	25	30	37

$f^* = 38$

* Shortest Path - Floyd Algorithm

Algorithm:

```
function Floyd(L(1...n, 1...n)) : array[1..n, 1...n]
        array D[1...n, 1...n]
        D ← L
        for K ← 1 to n do
            for i ← 1 to n do
                for j ← 1 to n do
                    D[i, j] ← min(D[i, j], D[i, K] + D[K, j])
        return D
```

↳ We construct a matrix D that gives the length of shortest path between each pair of nodes.

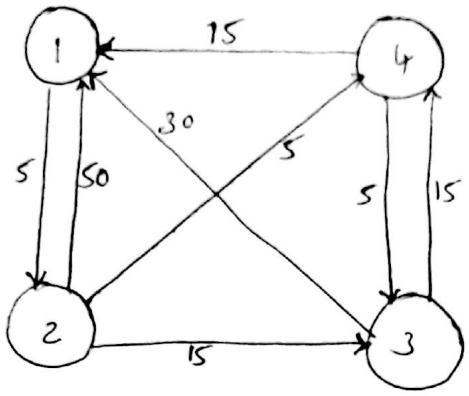
↳ The algorithm initializes D to L, that is to the direct distances between nodes. It then does n iteration after iteration K, D gives length of the shortest paths that only use nodes in (1, 2, ... K) as intermediate nodes.

↳ After n iterations, D therefore gives the length of shortest paths using any of the nodes in N as an intermediate node.

↳ If D_K represents the matrix D after K^{th} interation it can be implemented by $D_K[i, j] = \min(D_{K-1}[i, j], D_{K-1}[i, K] + D_{K-1}[K, j])$

↳ We use principle of optimality to compute length from i to j passing through K.

* Example:- Solve using Bell-man Floyd Algorithm.



Solution:-

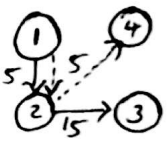
$$D_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix} \end{array}$$

- Create matrix - D_0.
- Node-1 is starting Node.

∴ 

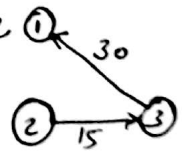
Add minimum distance in $D_0(1,2) = 5$

$$D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \underline{20} & \underline{10} \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

↳ In D_1 → Node 

∴ Add $D_1(1,3) = 20$.
Add $D_1(1,4) = 10$

$$D_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \underline{20} & \underline{10} \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

↳ In D_2 → Node 

update & Add $D_3(2,1) = 45$

$$D_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & 20 & 10 \\ \underline{45} & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

Similarly in matrix D_4.

$$D_4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \underline{15} & 10 \\ \underline{20} & 0 & \underline{10} & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

Matrix Chain Multiplication:

↳ Input: A chain of matrices to be multiplied

↳ output: A parenthesizing of the chain.

objective: Minimize number of steps needed for the multiplication

↳ Suppose we have a sequence or chain $A_1, A_2, \ldots, A_n$ of n matrices to be multiplied.

That is, we want to compute the product $A_1 A_2 \ldots A_n$

↳ There are many possible ways (Parenthesization) to compute the product.

↳ For Example: Consider the chain A_1, A_2, A_3, A_4 of 4 matrices.

- Let us compute the product $A_1 A_2 A_3 A_4$

5 different orderings = 5 different parenthesizations

1. $(A_1 (A_2 (A_3 A_4)))$
2. $(A_1 ((A_2 A_3) A_4))$
3. $((A_1 A_2)(A_3 A_4))$
4. $((A_1 (A_2 A_3) A_4)$
5. $(((A_1 A_2) A_3) A_4)$

- Matrix multiplication is associative,
  e.g., $A_1 A_2 A_3 = (A_1 A_2) A_3 = A_1 (A_2 A_3)$)

So Parenthesization does not change result.

- To compute the number of scalar multiplications necessary, we must know:
  - ↳ Algorithm to multiply two matrices
  - ↳ Matrix dimensions.

* Algorithm:-

```
Matrix chain order (P)
n ← length[P] - 1
for i ← 1 to n
    do m[i, i] ← 0
for l ← 2 to n      {: l is the chain length}
    do for i ← 1 to n - l + 1
        do j ← i + l - 1
           m[i, j] ← ∞
           for k ← i to j - 1
           do q ← m[i, k] + m[k+1, j], P_{i-1} P_k P_j
           if q < m[i, j]
              then [m[i, j] ← q.
                   s[i, j] ← k .
return m and s
```

↳ The time to compute C is dominated by the number of scalar multiplications.

↳ To illustrate the different costs incurred by different parenthesization of matrix product.

Matrix chain Multiplication using dynamic programming.

↳ No. of Multiplication required to multiply two matrices.

↳ For Example :- 1.

$M_1 = A_{5\times4}$ $M_2 = B_{4\times2}$.

$C_{5\times2} = A_{5\times4} \times B_{4\times2}$.

No. of multiplication required $= 5 \times 4 \times 2$.

$= 40$.

For Example: 2. $A_{10\times100} \times B_{100\times5} \times C_{5\times50} = D_{10\times50}$.

↳ Here there are two type of multiplication. exist.

1: $D = \overbrace{(A\ B)}^{X} C$

No. of Multiplications $= (10 \times 100 \times 5) + \underset{\text{mult}}{\text{No. of}} (X_{10\times5} \times C_{5\times50})$

$= (10 \times 100 \times 5) + (10 \times 5 \times 50)$

$= 7500$

2: $D = A \underbrace{(B\ C)}_{X}$

No. of multiplications $= (100 \times 5 \times 50) + \underset{\text{mult.}}{\text{No. of}} (A_{10\times100} \times X_{100\times50})$

$= (100 \times 5 \times 50) + (10 \times 100 \times 50)$

$= 75,000)$

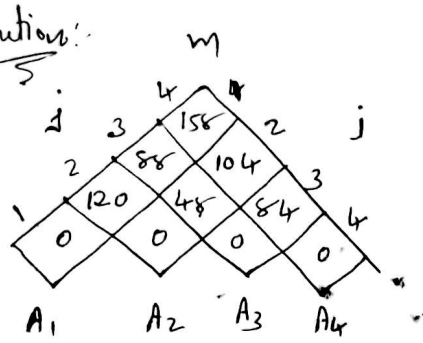
↳ Given a chain $A_1, A_2, \ldots A_n$ of n matrices, where for $i = 1, 2, \ldots n$, matrix A_i has dimension $p_{i-1} \times p_i$

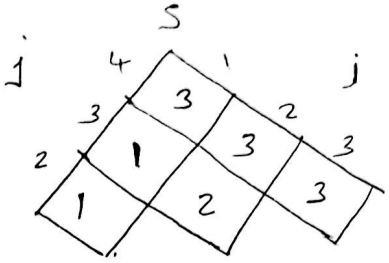
↳ Parenthesize the product $A_1, A_2, \ldots A_n$ such that the total no. of Scalar multiplication is minimized.

Example: 1. Consider the production.

$(P_0, P_1, P_2, P_3, P_4) = (5, 4, 6, 2, 7)$

Solution:





$((A_1)(A_2\ A_3)\ A_4)$

Step-1. $m[j, i] \leftarrow 0$ where $j = i$

Step-2. $m[1,2] = 0 + 0 + 5 \times 4 \times 6 = 120$

$m[2,3] = 0 + 0 + 4 \times 6 \times 2 = 48$.

$m[3,4] = 0 + 0 + 6 \times 2 \times 7 = 84$.

Step-3.

$$M[1,3] = \min \begin{cases} m[1,2] + m[3,3] + P_0 P_1 P_3 = 0 + 48 + (5 \times 4 \times 2) = 88 & \leftarrow k=1 \\ m[1,1] + m[2,3] + P_0 P_2 P_3 = 120 + 0 + (5 \times 6 \times 2) = 180 & \leftarrow k=2 \end{cases}$$

$= 88$

$$M[2,4] = \min \begin{cases} m[2,3] + m[4,4] + P_1 P_3 P_4 = 48 + 0 + (4 \times 2 \times 7) = 104 & \leftarrow k=1 \\ m[2,2] + m[3,4] + P_1 P_2 P_4 = 0 + 84 + (4 \times 6 \times 7) = 252 & \leftarrow k=2 \end{cases}$$

$= 104$

$$m[1,4] = \min \begin{cases} m[1,1] + m[2][4] + P_0 P_1 P_4 = 0 + 104 + (5 \times 4 \times 7) = 280 & \leftarrow k=1 \\ m[1,2] + m[3,4] + P_0 P_2 P_4 = 120 + 84 + (5 \times 6 \times 7) = 414 & \leftarrow k=2 \\ m[1,3] + m[4,4] + P_0 P_3 P_4 = 88 + 0 + (5 \times 2 \times 7) = 158 & \leftarrow k=3 \end{cases}$$

$= 158$.

Example:- Consider the following matrix.

$A_1[30 \times 35]$, $A_2[35 \times 15]$, $A_3[15 \times 5]$, $A_4[5 \times 10]$, $A_5[10 \times 20]$ and $A_6[20 \times 25]$ Solve using dynamic programming.

Solution.

M

i \ j	1	2	3	4	5	6
1	0	15,750	7,875	9,375	11,875	15,125
2		0	2,625	4,375	7,125	10,500
3			0	750	2,500	5,375
4				0	1,000	3,500
5					0	5,000
6						0

A_1 A_2 A_3 A_4 A_5 A_6

S

i \ j	1	2	3	4	5	6
1	1	2	3	4	5	5
2		1	3	3	5	5
3			3	3	3	4
4				3	3	3
5					3	3
6						1

$(((A_1)(A_2, A_3))(A_4, A_5)\ A_6)$

Chain order $n = 6$.

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + P_1 P_2 P_5 = 0 + 2500 + 35 \times 15 \times 20 = 13000 \\ m[2,3] + m[4,5] + P_1 P_3 P_5 = 2625 + 1000 + 35 \times 5 \times 20 = 7125 \\ m[2,4] + m[5,5] + P_1 P_4 P_5 = 4375 + 0 + 35 \times 10 \times 20 = 11374 \end{cases}$$

$\therefore m[2,5] = \min\{7125\}$

here, Similarly Solve the step up to $m[1,6]$...

Longest chain subsequence using dynamic programming.

Algorithm:-

```
LCS - LENGTH(x, y)
    m ← length(x)
    n ← length(y)
    for i ← 1 to m
        do c[i, 0] ← 0
    for j ← 1 to m
        do for i ← 1 to n
            do if xi = yj
                then c[i, j] ← c[i-1, j-1] + 1
                     b[i, j] ← "↖"
                else if c[i-1, j] ≥ c[i, j-1]
                    then c[i, j] ← c[i-1, j]
                         b[i, j] ← "↑"
                    else c[i, j] ← c[i, j-1]
                         b[i, j] ← "←"
    return c & b.
```

↳ We need to generate table c[1...m, 1...n] where m = length of string s_1 & n = length of string s_2.

↳ b[i][j] stores directions like (←, ↑, ↖)

↳ To generate table c[i][j] use following steps.

Step-1. Make c[i][0] = 0 & c[0][j] = 0.

Step-2. if $x_i = y_i$ then

c[i, j] ← c[i-1, j-1] + 1 & b[i, j] ← "↖"

Step-3. else if $c[i-1, j] \geq c[i, j-1]$

then $c[i, j] \leftarrow c[i-1, j]$ & $b[i, j] \leftarrow$ "↑"

Step-4. else $c[i, j] \leftarrow c[i, j-1]$ & $b[i, j] \leftarrow$ "←"

Example:- Find any one Longest Common Subsequence of given two strings using dynamic programming.

$S_1 = a\ b\ c\ d\ c\ d\ a\ b$ $S_2 = d\ c\ b\ c\ a\ a$.

Solution:-

		a	b	c	d	c	d	a	b
	0	1	2	3	4	5	6	7	8
d	0	0	0	0	0	0	0	0	0
d	1	0	0	0	0	1	1	1	1
b	2	0	0	0	1	1	2	2	2
b	3	0	0	1	1	1	2	2	2
a	4	0	0	1	2	2	2	2	3
a	5	0	1	1	2	2	2	3	3
a	6	0	1	1	2	2	2	3	3

b b a

↳ Find Longest Common Subsequence

- Start bottom Right Corner.

∴ Lcs = b c a.

Student Name EN. No. Contact. Reason

1 1 2 5 10 20 50

0/1 Knapsack Problem

V(1,...n, 0...w) where n = no. of objects $n = 5$.

w = capacity of knapsack $w = 11$

w_1 v_1
w_2 v_5
w_3
w_4
w_5

Step-1. make $V[i][0] = 0$ for $0 < i \leq n$

Step-2 if $j < w_i$ then take $V[i][j] = V[i-1][j]$

Step-3 if $j \geq w_i$ then take

$V[i][j] = \max(V[i-1][j], V[i-1][j-w_i] + v_i)$

① $V[0][j] = 0$ where
$j = 0$ to w, $w = 11$
$\therefore V[0][0] = 0 \ldots V[0][11] = 0$

② $V[i][0] = 0$ where ...

$V[1][1]$ $i = 1$, $j = 1$
$w_1 = 1$, $v_1 = 1$.

$j > w_i$

$V[1][1] = \max(V[1-1][1], V[1-1][1-1] + v_1)$
$= \max(0, 1+0)$
$= 1$

③ $V[1][2] = \max(V[0][2], V[0][1] + v_1)$
$= \max(0, 0+1)$
$= 1$

④ $V[2][1]$, $i = 2$, $j = 1$, $w_2 = 2$ $v_2 = 6$
$j < w_2$.
$V[2][1] = V[i-1][j]$
$= V[1][1]$
$= 1$

V.I ↓ / j →	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
$w_1=1$ $v_1=1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2=2$ $v_2=6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3=5$ $v_3=18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4=6$ $v_4=22$	0	1	6	7	7	18	22	24	28	29	29	40
$w_5=7$, $v_5=28$	0	1	6	7	7	18	22	28	29	34	35	40

[5][11]

4. (22)
5 (18)
2 (6)
1+6+18 + 22 + 28 (75)
40

6 (22) 6 22
5 (18) 5 18
11 40 11 40