UNIT-III

DYNAMIC PROGRAMMING

General Method, applications-optimal binary search trees, 0/1 Knapsack problem, All pairs shortest path problem, Traveling sales person problem, Reliability design.

Introduction:

Dynamic programming is typically applied to optimization problem. This technique is invented by a U.S. Mathematician Richard Bellman in 1950. In the word dynamic programming the word programming stands for planning and it does not mean by computer programming.

- ⊛ Dynamic programming is technique for solving problems with overlapping subproblems.
- ⊛ In this method each subproblem is solved only once. The result of each subproblem is recorded in a table from which we can obtain a solution to the original method.

General Method :-

⊛ Dynamic programming is typically applied to optimization problems.

For each given problem, we may get any number of solutions we seek for optimum solution (i.e., minimum value or maximum value). And such an optimal solution becomes the $\overset{sol^n}{\text{solution}}$ to the given problem.

Difference between Divide and Conquer and Dynamic programming.

<u>Divide and Conquer</u>	<u>Dynamic programming</u>
① The problem is divided in to small subproblems. These subproblems are solved independently. Finally all the solutions of subproblems are collected together to get the solution to the given problem.	① In dynamic programming many decision sequences are generated and all the overlapping subinstances are considered.
② In this method duplications in subsolutions are neglected i.e., duplicate subsolutions may be obtained.	② In dynamic programming Computing duplications in solutions is avoided totally.
③ Divide and Conquer is less efficient because of rework on solutions	③ It is efficient than divide and Conquer strategy.
④ This method uses top down approach of problem solving (recurssive methods)	④ Dynamic programming uses bottom up approach of problem solving (Iterative method)
⑤ Divide and Conquer splits its input at specific deterministic points usually in the middle	⑤ Dynamic programming splits its input at every possible split points rather than at a particular point. After trying all split points it determines which split point is optimal.

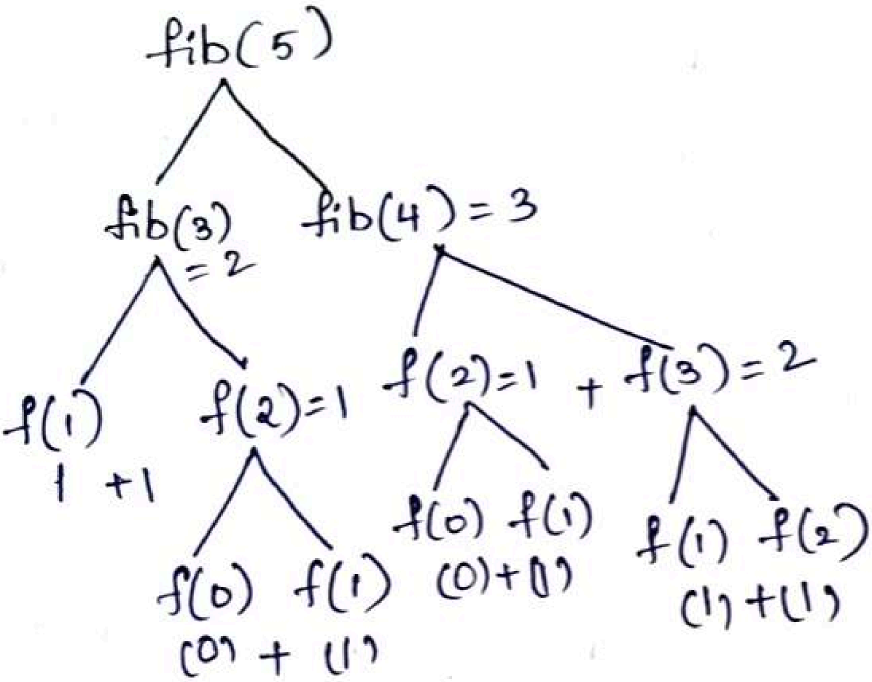
Example:- Fibonacci series.

Algorithm:-

$$fib(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \end{cases}$$

```
int fib(n)
{
   if (n<=1)
      return n;
   return fib(n-2)+fib(n-1)
}
```

0, 1, 1, 2, 3, 5, 8, 13, . . .



15 times we are calculating instead of it by using dynamic programming weare storing the values in table

0	1	2	3	4	5
x	x	x	x	x	x

0	1	2	3	4	5
0	1	1	2	3	5

Applications of Dynamic programming:

① optimal Binary search trees.

② 0/1 Knapsack problem

③ All pairs shortest path problem

④ Travelling salesperson problem

⑤ Reliability design

① Optimal Binary Search Trees:-

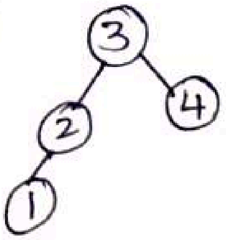
Binary tree → elements less than root, left side
greater than root, right side

optimal → for given set of numbers, we can build
multiple binary trees.

But, among all minimum number of steps to reach the key is called optimal Binary search Trees (OBST)

Dynamic programming: Big problem is broken into subproblems and each problem is solved individually

{1, 2, 3, 4}
↓
root



→ Each tree is represented as Ton

Ex: $n = 4$, $(a_1, a_2, a_3, a_4) = (\overset{a}{do}, \overset{b}{if}, \overset{c}{int}, \overset{d}{while})$

Successful search $P(1:4) = (3, 3, 1, 1)$

unsuccessful search $Q(0:4) = (2, 3, 1, 1, 1)$

p and q are probabilities

$w(i,j) = w(i,j-1) + p(j) + q(j)$

$c(i,j) = \min\limits_{i < k \leq j} \{c(i,k-1) + c(k,j)\} + w(i,j)$

$n = 4 \Rightarrow T_{04}$

$r(i,j) = k$

initial Conditions

$w(i,i) = q_i$

$c(i,i) = 0$

$r(i,i) = 0$

$j-i=0, j-i=1, j-i=2, j-i=3, j-i=4$

case 1: $j-i=0$, $\underbrace{T_{00}, T_{11}, T_{22}, T_{33}, T_{44}}_{w(i,j),\, c(i,j),\, r(i,j)}$ (T_{ij})

$T_{00} \rightarrow c(0,0) = 0$

$r(0,0) = 0$

$w(0,0) = q_0 = 2$.

$T_{11} \rightarrow c(1,1) = 0$

$r(1,1) = 0$

$w(1,1) = q_1 = 3$

$T_{22} \rightarrow c(2,2) = 0$, $r(2,2) = 0$ $w(2,2) = q_2 = 1$

$T_{33} \rightarrow c(3,3) = 0$, $r(3,3) = 0$ $w(3,3) = q_3 = 1$

$T_{44} \Rightarrow c(4,4) = 0$, $r(4,4) = 0$ $w(4,4) = q_4 = 1$

Write all the values

T_{00}	T_{11}	T_{22}	T_{33}	T_{44}
$C(0,0)=0$	$C(1,1)=0$	$C(2,2)=0$	$C(3,3)=0$	$C(4,4)=0$
$r(0,0)=0$	$r(1,1)=0$	$r(2,2)=0$	$r(3,3)=0$	$r(4,4)=0$
$w(0,0)=2$	$w(1,1)=3$	$w(2,2)=1$	$w(3,3)=1$	$w(4,4)=1$

Case2: $j-i=1 \Rightarrow T_{01}, T_{12}, T_{23}, T_{34}$

for T_{01}, $w(0,1) = w(0,0) + p(1) + q(1)$

$$= 2+3+3=8$$

$$w(0,1)=8$$

$$C(0,1) = \min_{\substack{0<K\leq 1 \\ K=1}} \{c(0,0)+c(1,1)\}+w(0,1)$$

$$c(0,1) = 0+0+8=8$$

$$r(i,j)=K=1$$

for T_{12} , $w(1,2) = w(1,1) + P(2) + q(2)$

$$= 3+3+1=7$$

$$c(1,2) = \min_{\substack{1<K\leq 2 \\ K=2}} \{c(1,1)+c(2,2)\}+w(1,2)$$

$$c(1,2) = 0+0+7=7$$

$$r(1,2)=K=2$$

for T_{23} , $w(2,3) = w(2,2) + p(3) + q(3)$

$$= 1+1+1=3$$

$$c(2,3) = \min_{\substack{2<K\leq 3 \\ K=3}} \{c(2,2)+c(3,3)\}+w(2,3)$$

$$= 0+0+3=3$$

$$r(2,3)=3$$

For $T_{34} :\Rightarrow w(3,4) = w(3,3) + p(4) + q(4)$

$$= 0 + 1 + 1 = 3$$

$$w(3,4) = 3$$

$$c(3,4) = \min_{3 < K \le 4} \{c(3,3) + c(4,4)\} + w(3,4)$$

$K = 4$

$$= \{0 + 0\} + 3 = 3$$

$$r(3,4) = K = 4$$

Case: 2 $j - i = 1$

T_{01}	T_{12}	T_{23}	T_{34}
$c(0,1) = 8$	$c(1,2) = 7$	$c(2,3) = 3$	$c(3,4) = 3$
$r(0,1) = 1$	$r(1,2) = 2$	$r(2,3) = 3$	$r(3,4) = 4$
$w(0,1) = 8$	$w(1,2) = 7$	$w(2,3) = 3$	$w(3,4) = 3$

Case 3: $j - i = 2$

T_{02}	T_{13}	T_{24}
$c(0,2) = 19$	$c(1,3) = 12$	$c(2,4) = 8$
$r(0,2) = 1$	$r(1,3) = 2$	$r(2,4) = 3$
$w(0,2) = 12$	$w(1,3) = 9$	$w(2,4) = 5$

Case 3, where $j - i = 2 \Rightarrow T_{02}, T_{13}, T_{24}$

$T_{02} \rightarrow w(0,2) = w(0,1) + p(2) + q(2)$

$$= 8 + 3 + 1 = 12$$

$$w(0,2) = 12$$

$$c(0,2) = \min_{0 < K \le 2} \begin{Bmatrix} K=1, c(0,0) + c(1,2) \\ K=2, c(0,1) + c(2,2) \end{Bmatrix} + w(0,2)$$

$K = 1, 2$

$$= \min \begin{Bmatrix} \overset{\checkmark}{K=1}, \ 0 + 7 = 7 \\ K=2, \ 8 + 0 = 8 \end{Bmatrix} + 12$$

$$c(0,2) = 7 + 12 = 19$$

$$r(0,2) = K = 1$$

$T_{13} \Rightarrow w(1,3) = w(1,2) + P(3) + q(3)$

$= 7+1+1 = 9$

$w(1,3) = 9$

$$C(1,3) = \min_{\substack{1<K\leq 3 \\ K=2,3}} \begin{Bmatrix} K=2,\ C(1,1)+C(3,3) \\ K=3,\ C(1,2)+C(3,3) \end{Bmatrix} + w(1,3)$$

$$= \min \begin{Bmatrix} 0+\overset{\checkmark}{3} \\ 7+0 \end{Bmatrix} + 9$$

$C(1,3) = 3+9 = 12$

$r(1,3) = K = 2$

$T_{24} \rightarrow w(2,4) = 5$

$C(2,4) = 8$

$r(2,4) = 3$

case 4: $j-i = 3 \rightarrow T_{03}, T_{14}$

$T_{03} = w(0,3) = w(0,2) + P(3) + q(3)$

$= 12+1+1 = 14$

$w(0,3) = 14$

$$C(0,3) = \min_{\substack{0<K\leq 3 \\ K=1,2,3}} \begin{Bmatrix} K=1,\ C(0,0)+C(1,3) \\ K=2,\ C(0,1)+C(2,3) \\ K=3,\ C(0,2)+C(3,3) \end{Bmatrix} + w(0,3)$$

$$= \min \begin{Bmatrix} K=1,\ 0+12 \\ K=2,\ 8+3\checkmark \\ K=3,\ 19+0 \end{Bmatrix} + 14$$

$C(0,3) = 11+14 = 25$

$r(0,3) = K = 2$

$T_{14} \Rightarrow w(1,4)=11$

$c(1,4)=19$

$r(1,4)=2$

Case 5 $\Rightarrow j-i=4 \Rightarrow T_{04}$

$T_{04} = w(0,4),\ c(0,4),\ r(0,4)$

$w(0,4) = w(0,3)+p(4)+q(4)$

$= 14+1+1=16$

$w(0,4)=16$

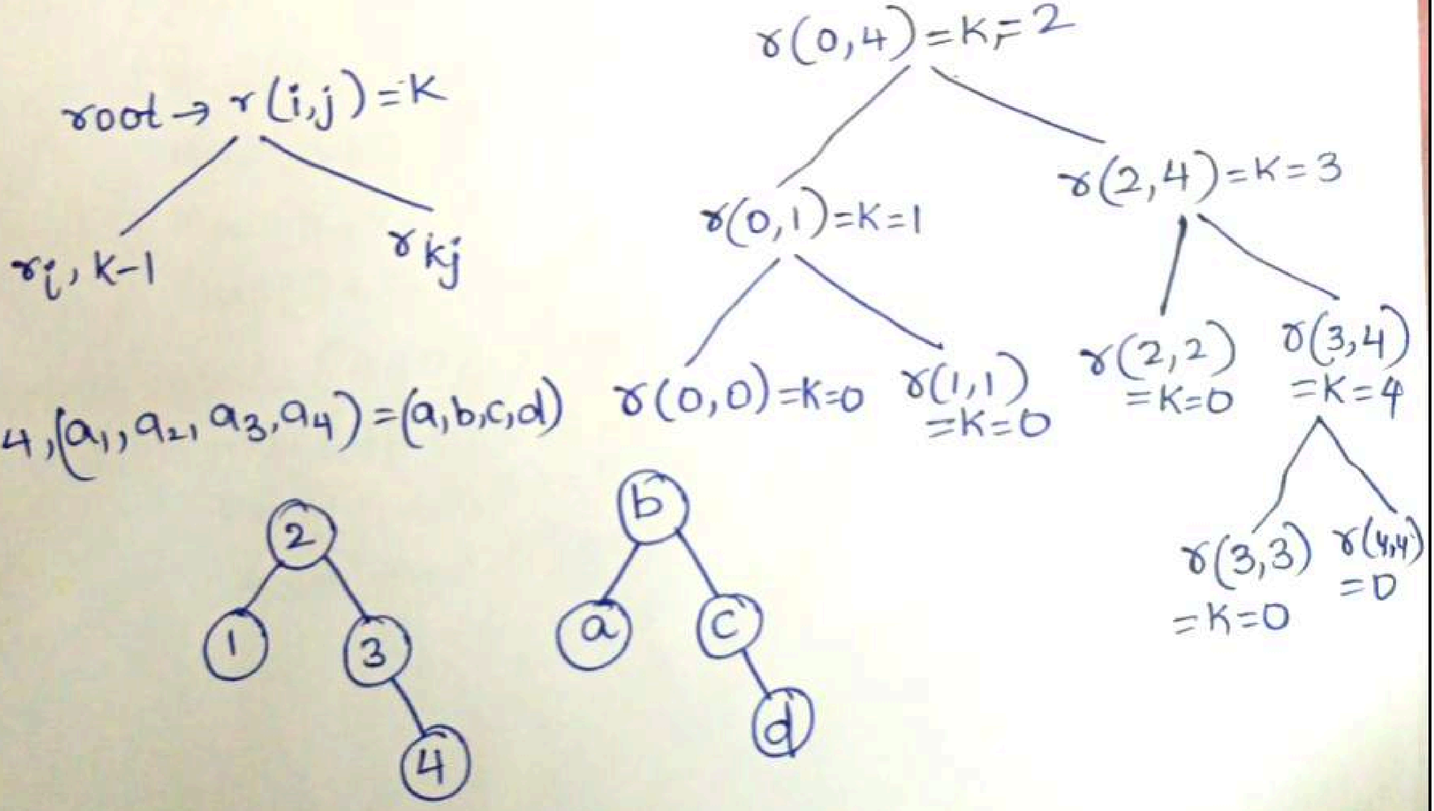
$$c(0,4) = \min_{\substack{0<K\le 4 \\ K=1,2,3,4}} \left\{ \begin{array}{l} K=1,\ C(0,0)+C(1,4) \\ K=2,\ C(0,1)+C(2,4) \\ K=3,\ C(0,2)+C(3,4) \\ K=4,\ C(0,3)+C(4,4) \end{array} \right\} + w(0,4)$$

$$= \min \left\{ \begin{array}{l} K=1,\ 0+19 \\ K=2,\ 8+5 ✓ \\ K=3,\ 19+3 \\ K=4,\ 25+0 \end{array} \right\} + 16$$

$c(0,4)= 13+16 = 29$

$r(0,4)=K=2$

$n=4, (a_1, a_2, a_3, a_4) = (a, b, c, d)$



*0/1 Knapsack problem using Dynamic programming

0/1 → either you pick the element/item completely or you don't pick them at all.
(no splitting)

Example :-
weights = {3, 4, 5, 6} and profit = {2, 3, 4, 1}
total weight = 8 and total items (n) = 4

profit (P_i)	weight (w_i)	weight →	0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
2	3	1	0	0	0	2	2	2	2	2	2
3	4	2	0	0	0	2	3	3	3	5	5
4	5	3	0	0	0	2	3	4	4	5	6
1	6	4	0	0	0	2	3	4	4	5	6

Profit → Max(3+0, 2) = max (3, 2) → 3
Max(3+0, 2) = max (3, 2) → 3
Max(3+0, 2) = 3
Max(3+2, 2) = 5
Max(3+2, 2) = 5

profit → max(4+0, 3) = 4
max(4+0, 3) = 4
Max(4+0, 5) = 5
Max(4+2, 5) = 6

profit → Max(1+0, 4) = 4
max(1+0, 5) = 5
Max(1+0, 6) = 6

{–, –, 1, –}

6 - 4 = 2

⊛ All Pairs shortest path problem:-

The All-pairs shortest path (APSP) problem is about finding the shortest path between every pair of vertices in a weighted graph, directed graph.

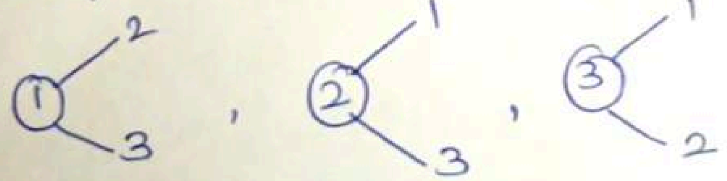
→ Dynamic programming provides an effective way to solve this, most notably with the Floyd-Warshall algorithm. This algorithm systematically improves its estimates of shortest path by considering an increasing number of intermediate vertices.

The Floyd-Warshall Algorithm:-

This algorithm works by iteratively updating a distance matrix that initially holds the direct edge weights. It uses dynamic programming to solve the problem by considering all possible intermediate vertices for each pair of start and end vertices. The core idea is that the shortest path from vertex i to j either passes through a new intermediate vertex, k, or it doesn't.

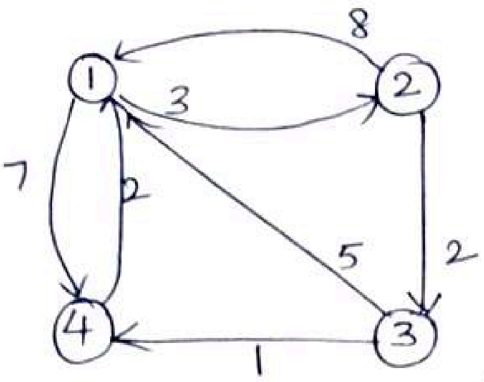
→ We are finding the shortest path b/w every pair of vertices.

Example → 3 vertices → {1, 2, 3}



→ Matrices are used to solve these problems

Example problem:-



① Create a distance matrix, M^0, of size V×V, where is the number of vertices .

$M^0[i][j] = w(i,j)$ if there's an edge from i to j

$M^0[i][j] = \infty$ if there's no direct edge.

$M^0[i][i] = 0$ for all vertices i.

M^0 → original matrix

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 7 \\ 2 & 8 & 0 & 2 & \infty \\ 3 & 5 & \infty & 0 & 1 \\ 4 & 2 & \infty & \infty & 0 \end{array}$$

M' [vertex 1] → 1st row, 1st column same from previous matrix, diagonal is 0

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 7 \\ 2 & 8 & 0 & 2 & 15 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & \infty & 0 \end{array}$$

$$M^0[2,3] = M^0[2,1] + M^0[1,3]$$

$$\underset{\checkmark}{2} = 8 + \infty$$

$$M^0[2,4] = M^0[2,1] + M^0[1,4]$$

$$\infty = 8 + 7 = 15 \checkmark$$

$M^0[3,2] = M^0[3,1] + M^0[1,2]$

$\infty \quad = 5+3$

$= 8$ ✓

$M^0[3,4] = M^0[3,1] + M^0[1,4]$

$\overset{}{1}$ ✓ $\quad = 5+7$

$= 12$

$M^0[4,2] = M^0[4,1] + M^0[1,2]$

$\infty \quad = 2+3$ ✓

$= 5$

$M^0[4,3] = M^0[4,1] + M[1,3]$

$\infty = 2+\infty$

M^2 (vertex 2) → 2nd Row, 2nd column, diagonal same from previous

$$\Rightarrow \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 7 \\ 2 & 8 & 0 & 2 & 15 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array}$$

$M'[3,4] = M'[3,2] + M'[2,4]$

$1 < 8+15\,(23)$

$M^1[1,3] = M^1[1,2] \overset{M'}{+} [2,3]$

$\infty \quad = 3+2$

$= 5$ ✓

$M^1[1,4] = M'[1,2] \overset{M'}{+} [2,4]$

$\overset{✓}{7} \quad = 3+15$

$M^1[3,1] = M'[3,2] + M'[2,1]$

$= 8+8$

$5 \quad = 16$

$M'[4,1] = M'[4,2] + M'[2,1]$

2 ✓ $= 2+8$

$M'[4,3] = M'[4,2] + M'[2,3]$

$\infty \quad = 5+2$

$= 7$ ✓

M

M^3 (vertex 3) → 3^{rd} row & 3^{rd} colum same from previous matrix

$$M^3 \Rightarrow \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 6 \\ 2 & 7 & 0 & 2 & 3 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array}$$

$M^2[1,2] = M^2[1,3] + M^2[3,2]$

$3^{\checkmark} = 5+8$

$M^2[1,4] = M^2[1,3] + M^2[3,4]$

$7 = 5+1$

$7 \ngeq 6^{\checkmark}$

$M^2[2,1] = M^2[2,3] + M^2[3,1]$

$8 = 2+5$

$= 7^{\checkmark}$

$M^2[2,4] = M^2[2,3] + M^2[3,4]$

$15 = 2+1$

M^4 (vertex 4) → 4^{th} Row & 4^{th} column same from M^3 matrix & diagonal same

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 6 \\ 2 & 5 & 0 & 2 & 3 \\ 3 & 3 & 6 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array}$$

$M^3(1,2) = M^3(1,4) + M^3(4,2)$

$= 6+5$

General formulae :

$$A^{K}[i,j] = \min\{A^{K-1}[i,j],\ A^{K-1}[i,K] + A^{K-1}[K,j]\}$$

$$M^{4}[2,3] = \min\{M^{3}[2,3],\ M^{3}[2,4] + M^{3}[4,3]\}$$

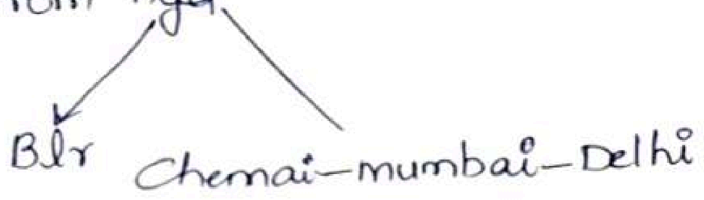
(*) Traveling * Sales * Person * problem :

→ Salesman will travel all the given cities and will Come back to city he started.

→ The Travelling Salesperson problem (TSP) using dynamic programming aims to find the shortest possible route that visits every city exactly once and returns to the starting city. This apporach leverages the principle of optimal Substructure and overlapping subproblems inherent in dynamic programming.

Ex: (hyd, Blr, chennai, Mumbai, Delhi)

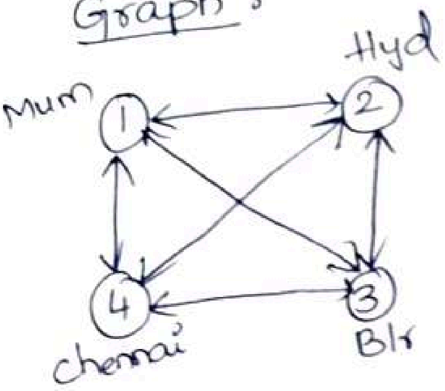
started from hyd



Should choose path with minimum cost

Example problem:-

Graph:



Cost Matrix

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 10 & 15 & 20 \\ 2 & 5 & 0 & 9 & 10 \\ 3 & 6 & 13 & 0 & 12 \\ 4 & 8 & 8 & 9 & 0 \end{array}$$

General formulae:-

$$g(i,S) = \min_{K \in S} \{ C_{iK} + g(K, S-\{K\}) \}$$

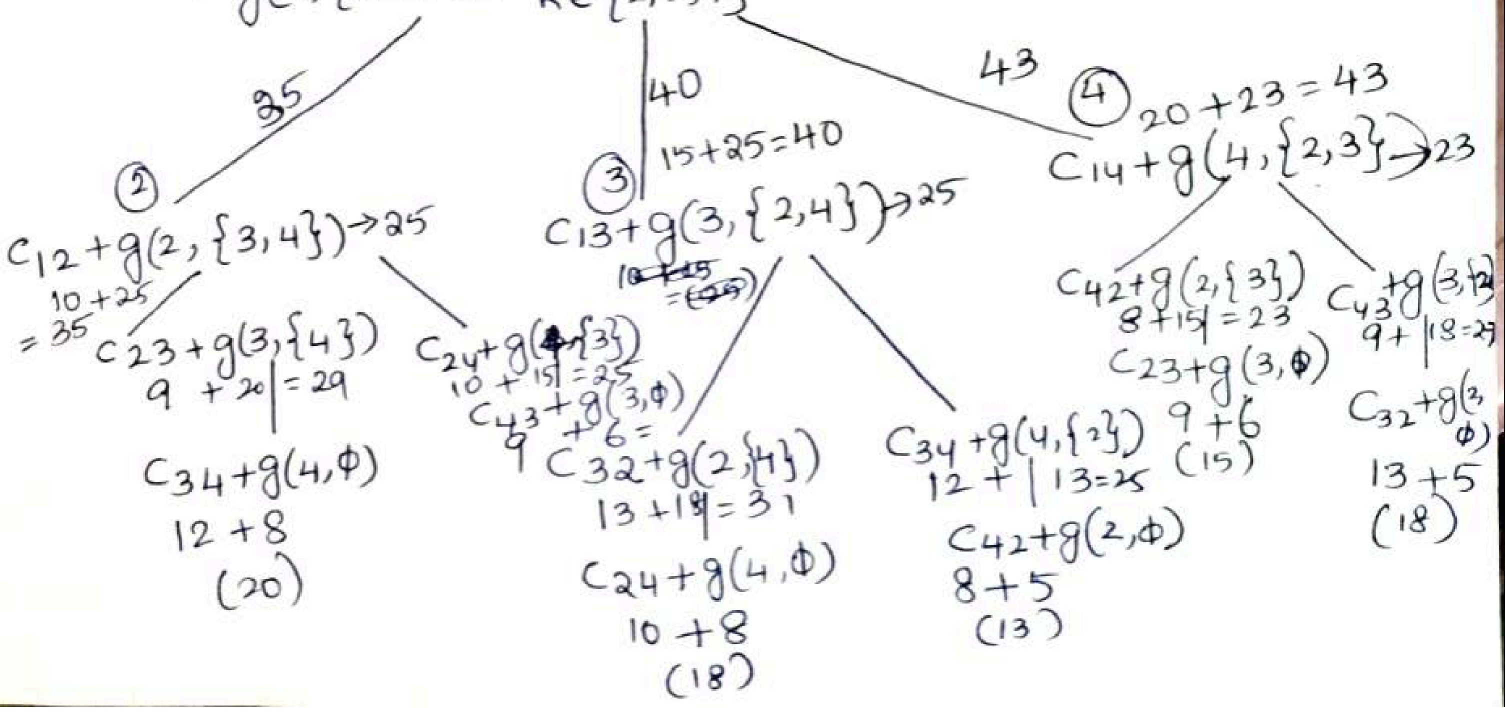
$$① \xrightarrow{10} 2 \xrightarrow{10} 4 \xrightarrow{9} 3 \xrightarrow{6} ①$$
$$\Rightarrow 35$$

$i \rightarrow$ initial vertex

$S \rightarrow$ set of remaining vertices

Starting from ①

$$g(1,\{2,3,4\}) = \min_{K \in \{2,3,4\}} \{ C_{1K} + g(K, \{2,3,4\} - \{K\}) \}$$



5 Reliability Design :-

Reliability design in dynamic programming addresses the problem of maximizing the overall reliability of a system, typically composed of multiple stages or devices, within a given cost constraint. This approach is particularly useful when individual components have varying costs and reliabilities, and the system's overall reliability depends on the functioning of all its parts.

Key aspects :-

System structure :- The system is often conceptualized as a series of stages, where each stage might contain multiple copies of a device connected in parallel to enhance its reliability. If one copy fails, others can take over, increasing the stage's overall reliability.

Reliability Calculation :-

→ For a single device with reliability 'R', having 'm' copies in parallel at a stage, the reliability of that stage is calculated as : $1-(1-R)^m$.

→ For a system with multiple stages in series, the overall system reliability is the product of the reliabilities of each individual stage.

Cost Constraint :-

Each device or copy of a device comes with a cost. The objective is to maximize the system's reliability while ensuring the total cost does not exceed a predefined budget.

Example:- Reliability → probability that the system/device/ requirement will work correctly.

I am setting up a network, requirements
1. Computers 2. Internet 3. Routers 4. Cables.

Reliability → Maximum
Cost → minimum.

⊛ Design a 3 stage system with device types D_1, D_2, D_3. The costs are 30, 15, 20. The cost of the systems is to be no more than 105. The reliabitity of each device type is 0.9, 0.8, 0.5.

	D_i	C_i	R_i	u_i
1 ←	D_1	30	0.9	2
2 ←	D_2	15	0.8	3
2 ←	D_3	20	0.5	3

D_i → Devices
C_i → Cost of each device
R_i → Reliabity of device
u_i → Upperbound.

Sol:- $C_1 = 30,\ C_2 = 15,\ C_3 = 20,\ C = 105$

$r_1 = 0.9,\ r_2 = 0.8,\ r_3 = 0.5$

Actual cost $= \Sigma C_i$
$= 30 + 15 + 20 = 65$

Remaining Cost ⇒ $105 - 65 = 40$

$$u_i = \text{floor}\left\lfloor \frac{C - \Sigma C_i}{C_i} \right\rfloor + 1$$

$$u_1 = \left[\frac{105-65}{30}\right] + 1 = \left[\frac{40}{30}\right] + 1 = (\overset{1}{1.33}) + 1 = 2$$

$$u_2 = \left[\frac{105-65}{15}\right] + 1 = \left[\frac{40}{15}\right] + 1 = (\overset{2}{2.5}) + 1 = 3$$

$$u_3 = \left[\frac{105-65}{20}\right] + 1 = \left(\frac{40}{20}\right) + 1 = 2 + 1 = 3$$

$S^0 = \{(1,0)\}$ $\qquad (R, C) = (1, 0)$

max R
min C

$D_1 \to 1$ copy $\Rightarrow (0.9, 30) = S_1^1 \leftarrow$ copy
$\qquad$ 1 $\leftarrow$ device

$(0.9, 30)$ $\quad$ 2 copy $\Rightarrow (0.99, 60) = S_2^1$ $\quad R = 1 - (1 - r_1)^2$

$= 1 - (1 - 0.9)^2$

$\Rightarrow 1 - (0.1)^2 = 1 - 0.01 = 0.99$

$S^1 = D_1 \Rightarrow \boxed{(0.9, 30), (0.99, 60)}$

$D_2 \to S_1^2 = 1$ copy $\Rightarrow (0.8 \times 0.9, 15 + 30), (0.8 \times 0.99 + 15 + 60)$

$(0.8, 15)$ $\qquad = \{(0.72, 45), (0.792, 75)\}$

$S_2^2 = 2$ copy $= (0.96 \times 0.9, 30 + 30), (0.96 \times 0.99, 30 + 60)$ $\quad 1 - (1 - 0.8)^2 = 1 - (0.2)^2$
$1 - 0.04 = 0.96$

$(0.96, 30)$

$= \{(0.864, 60), (0.9504, 90)\}$
$\qquad\qquad\qquad\qquad$ (X)

3 copies $= (0.992 \times 0.9, 45 + 30),$ $\quad (0.992 \times 0.99, 45 + 60)$
$\qquad\qquad\qquad\qquad\qquad\qquad$ (X)

$(0.992, 45)$

$1 - (1 - 0.8)^3$
$= 1 - (0.2)^3$
$1 - 0.008$
$= 0.992$

$= (0.8928, 75), (\qquad , 105)$

$D_2 \to (0.72, 45), \underset{\times}{(0.792, 75)}, (0.864, 60), (0.8928, 75)$

$\boxed{D_2 \to (0.72, 45), (0.864, 60), (0.8928, 75)}$

$D_3 \to 1$ copy $\to (0.5 \times 0.72, 20 + 45), (0.5 \times 0.864, 20 + 60),$
$(0.5 \times 0.8928, 20 + 75)$

$(0.5, 20)$

$S_1^3 \to (0.36, 65), (0.43, 80), (0.4464, 95)$
$\qquad\qquad$ 1 $\qquad\quad$ 2 $\qquad\quad$ 3

$1 - (1 - r_1)^2$
$= 1 - (1 - 0.5)^2$
$= 1 - (0.5)^2$
$= 0.75$

2 copies $\to (0.75 \times 0.~~72~~, 40 + 45), (0.864 \times 0.75, 40 + 60)$
$\underset{\times}{(0.75 \times 0.8928, \underbrace{75 + 40}_{115})}$

$(0.75, 40)$

$\Rightarrow (0.54, 85), (0.648, 100)$
$\qquad\quad$ 4 $\qquad\qquad$ 5

3 copies $\to (0.875 \times 0.72, 60 + 45), ~~60~~$

$(0.875, 60)$ $\quad \Rightarrow (0.63, 105)$
$\qquad\qquad\qquad\qquad$ 5

$1 - (1 - 0.5)^3$
$\Rightarrow 1 - (0.5)^3$
$\Rightarrow 1 - 0.125$
$= 0.875$

$D_3 \Rightarrow \{(0.36, 65), (0.432, 80), \underset{}{\overset{\times}{(0.4464, 95)}}, (0.54, 85),$
$\underset{\times}{(0.63, 105)}, (0.648, 100)\}$

Maximum Reliability ⇒ 0·648

Cost ⇒ 100

$D_1 \rightarrow$ 1 copy (30)

$D_2 \rightarrow$ 2 copy (30)

$D_3 \rightarrow$ 2 copy $\underline{(40)}$

$\underline{100}$