Module → 2 Fundamental Algorithmic Strategies :-

: Brute-force :: linear search, Selection Sort,

* Greedy : Huffman Coding, fractional Knapsack proble, Activity selection Problem.

* Dynamic Programming :- Matrix Chain multiplication, longest Common Subsequence, Travelling Salesman Problem, Branch-and-Bound

* Backtracking methodologies for the design of algorithms ; Illustrations of these techniques for Problem-Solving, Bin-Packing, Knap Sack TSP,

* Heuristics - characteristics and their application domains.

* **Dynamic Programming** :- Dynamic programming is a name, coined by Richard Bellman in (1955). Dynamic programming, as greedy method, is a powerful algorithm design technique that can be used when the solution to the problem may be viewed as the result of a sequence of decisions. In dynamic programming we examine the decision sequence to see whether an Optimal decision sequence contains Optimal decision subsequence.

⇒ When Optimal decision sequences Contain optimal decision subsequences, we can establish recurrence equations, Called dynamic-programming recurrence equations, that enable us to solve the problem in an efficient way.

⇒ We typically apply dynamic programming to Optimization problems. Such problem can have many possible solutions. Each solution has a value and we wish to find a solution with the Optimal (minimum and maximum) value. We call such a solution an Optimal sol^{n} to the problem, as opposed to the optimal solution, Since there may be several sol^{n} that achieve the Optimal value.

When developing a dynamic-programming algorithm, we follow a sequence of four steps:

1) Characterize the structure of an Optimal Sol^{n}.
2) Recursively define the value of an Optimal ".

P.T.O.

3.) Compute the value of an Optimal Soln, typically in a bottom-up fashion.

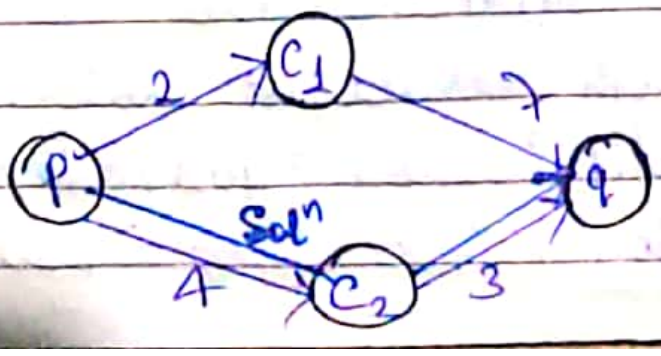
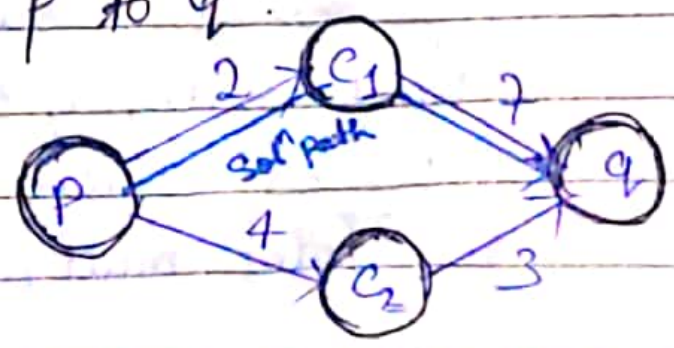
4.) Construct an Optimal soln from computed information.

Steps 1-3 form the basis of a dynamic-programming soln to a problem. If we need only the value of an Optimal soln, and not the soln itself, then we can Omit step 4. When we do perform step 4, we sometimes maintain additional info during step 3 so that we can easily construct an Optimal soln.

Short Note :-

- Dynamic programming is similar to "Divide And Conquer".
- It divides problem into number of Subproblems.
- Solution of a Subproblem is stored in a table for future use.

Dynamic programming	Divide and Conquer
• Subproblems are dependent.	• Sub-problems are Independent.
• Same Subproblem is not Calculated every time it is required.	• Same Subproblem is Calculated every time it is required.

Dynamic Programming	Greedy Programming
⇒ It is an algorithm design technique.	⇒ It is an algorithm design technique.
⇒ Suitable for Variety of Problems.	⇒ Suitable for Specific Problem.
⇒ Many decission sequence is Generated.	⇒ One decission sequence is Generated.
⇒ Bottom up approach.	⇒ Top down approach.
⇒ Example:- Longest Common Subsequence, Matrix Chain multiplication.	⇒ Example:- Huffman Coding Knapsack problem, Activity Selection problem
⊛ Elements of Dynamic Programming.	★ Elements of Greedy
1⇒ Optimal Substructure optimal Sol^n of problem = Combination of optimal Sol^n of all Subproblems	1⇒ Optimal Substructure No.
2⇒ Overlapping Subproblems Subproblems are Solved again and again.	2⇒ Greedy property Select the Sol^n path which looks best right now.
Ex:- Shortest path from "p" to "q". 	Ex:- Shortest path from "p" to "q". 

MCM (Matrix Chain Multiplication)

» Matrix chain means number of matrices.
» We want to multiply number of matrices
Each matrix has an order i.e Row x Column

Q.1) Consider 2 Matrices A_1, A_2.

$$\underset{2\times3}{A_1}\begin{bmatrix}3 & 6 & 4\\ 5 & 1 & 7\end{bmatrix}\quad \underset{3\times4}{A_2}\begin{bmatrix}2 & 6 & 5 & 8\\ 6 & 7 & 4 & 5\\ 9 & 3 & 2 & 4\end{bmatrix} = \underset{2\times4}{\begin{bmatrix}\cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot\end{bmatrix}}$$

(m x n) (n x p) (m x p)

Total no. of Scalar multiplication = m x n x p
= (2 x 3 x 4)
= 24.

Q.2) Consider 3 matrix A_1, A_2, A_3 and Order of $A_1 = 10 \times 7$ and Order of $A_2 = 7 \times 5$ & Order of $A_3 = 5 \times 20$. Consider the following Case of multiplication.

Case-1 :- $A_1 (A_2, A_3)$
Case-2 :- $A_3 (A_1, A_2)$ and which One is better after multiplication?

Case → 1 :- If A_2 and A_3 is multiplied then no. of multiplication (Scalar) = 7 x 5 x 20 = 700.

Order of resultant matrix = 7 x 20
Now multiply A_1 with resultant matrix.
No. of Scalar multiplication = 10 x 7 x 20 = 1400.
Total multiplication = 700 + 1400 = 2100.

Case-2 :- $(A_1, A_2)A_3$

If A_1 and A_2 is multiplied, then no. of Scalar - multiplication = $10 \times 7 \times 5 = 350$

Order of resultant matrix = $10 \times 5 =$

Now multiply A_3 with resultant matrix.

No. of Scalar multiplication = $10 \times 5 \times 20 = 1000$.

Total multiplication = $1000 + 350 = 1350$.

⇒ Since, the total multiplication of Case→2 is less so, Case 2 is better as Compared to Case→1 and Case→2 is the optimal soln to my problem.

* Example/Q.3→ Consider the following Case of multiplication:

Case 1 :- $(((A_1, A_2)A_3)A_4)$

Case 2 :- $((A_1, (A_2 A_3))A_4)$

Case 3 :- $(A_1(A_2(A_3, A_4)))$

Case 4 :- $((A_1 A_2)(A_3 A_4))$

⇒ Bracket is called parenthesis.

Bracket will decide the "Sequence of multiplic:"

Sequence will decide the "total no of Scalar multiplications".

→ The Case having minimum no of multiplications is called best Case (optimal Case). So, it is called optimal parenthesization.

Step → 1 :- ["Determine the structure of Optimal Parenthilation"] :-

Let us Consider a matrix A_i ---- A_j, where $i \leq j$.
Now the problem is non-trivial if $i < j$ then any Parenthization of the product bet^n A_K and A_{K+1} for some integer K and the range is $i \leq K \leq j$
i.e; for some K we first Compute A_i ---- A_K and A_{K+1} ---- A_j. multiply them together to get the final result.

Step → 2 ["Recursive Solution to MCM"]

⇒ Let $m[i,j]$ denotes minimum no. of Scalar multiplication for the sub Problem A_i ---- A_j
$m[1,n]$ denotes minimum no. of Scalar multiplication needed to Compute the whole problem A_1 ---- A_n.

⇒ If $i = j$, $m_{ij} = 0$ and the problem is A_i only.
⇒ If $i < j$, $m[i,j] = m[i,K] = m[i,K] + m[K+1, j] + P_{i-1} P_K P_j$

$$\Rightarrow m[i,j] = \begin{cases} 0 & ; i = j \\ m[i,K] + m[K+1,j] + P_{i-1} P_K P_j & ; i \leq K \leq j \end{cases}$$

Dimension of $A_i = P_{i-1} \times P_i$.
So, we define the recursive Sol^n,

$$m[i,j] = \begin{cases} 0 & \\ m[i,K] + m[K+1,j] + P_{i-1} P_K P_j & i \leq K \leq j \end{cases}$$

Let $S[i,j] = K$, if we split the Product $A_i, A_{i+1} \cdots A_j$ to obtain an Optimal Parathesization.

⊛ Matrix chain Order :- (P)

```
1. n ← length[P] - 1.
2. for i ← 1 to n.
3. do m[i,i] ← 0.
4. for l ← 2 to n.
5. do for i ← 1 to n-l+1.
6. j ← i+l-1.
7. m[i,j] ← ∞.
8. for K ← i to j-1.
9. do q ← m[i,K] + m[K+1,j] + P_{i-1} x P_j x P_K.
10. if q < m[i,j].
11. then m[i,j] ← q.
12. S[i,j] ← K.
13. Return m and S.
```

Q) Consider the following matrix and its Dimension.
here,

$m_1 = 5 \times 12$; $m_4 = 20 \times 5$

$m_2 = 12 \times 10$; $m_5 = 5 \times 30$; $m_7 = 10 \times 5$.

$m_3 = 10 \times 20$; $m_6 = 20 \times 10$

Sol^n ⇒ $P_0 = 5$; $P_1 = 12$; $P_2 = 10$; $P_3 = 20$; $P_4 = 5$; $P_5 = 30$; $P_6 = 10$; $P_7 = 5$.

$i \rightarrow$

j ↓	1	2	3	4	5	6	7
7	m_{17}	m_{27}	m_{37}	m_{47}	m_{57}	m_{67}	m_{77}
6	m_{16}	m_{26}	m_{36}	m_{46}	m_{56}	m_{66}	
5	m_{15}	m_{25}	m_{35}	m_{45}	m_{55}		
4	m_{14}	m_{24}	m_{34}	m_{44}			
3	m_{13}	m_{23}	m_{33}				
2	m_{12}	m_{22}					
1	m_{11}						

for Diagonal ($S = 0$)

⇒ $m[1,1] = m[2,2] = m[3,3] = m[4,4] = m[5,5] = m[6,6] = m[7,7] = 0$

for Diagonal ($S = 1$).

⇒ $m[i,j] = m[i,k] + m[k+1,j] + P_{i-1} \times P_j \times P_k$.

⇒ $m[1,2]$; ⇒ $k = i$ to $j-1 = 1$ to $2-1 = 1$.

⇒ $m[1,1] + m[2,2] + P_0 \times P_1 \times P_2 = 0 + 0 + 5 \times 12 \times 10 = 600$.

$m[1,2] = 600$; $k = 1$

$\Rightarrow m[2,3]; \Rightarrow K = i$ to $j-1 = 2$ to $3-1 = 2$.

$\Rightarrow m[2,2] + m[3,3] + P_1 \times P_2 \times P_3 = 0+0+12\times10\times20 = 2400$

$\boxed{m_1[2,3] = 2400;\ K=2.}$

$\Rightarrow m[3,4]; \Rightarrow K = i$ to $j-1 = 3$ to $4-1 = 3$.

$\Rightarrow m[3,3] + m[4,4] + P_2 \times P_3 \times P_4 = 0+0+10\times20\times5 = 1000.$

$\boxed{m[3,4] = 1000;\ K=3}$

$\Rightarrow m[4,5]; \Rightarrow K = i$ to $j-1 = 4$ to $4 = 4$.

$\Rightarrow m[4,4] + m[5,5] + P_3 \times P_4 \times P_5 = 0+0+20\times5\times30 = 3000$

$\boxed{m[4,5] = 3000;\ K=4}$

$\Rightarrow m[5,6]; \Rightarrow K = i$ to $j-1 = 5$ to $5 = 5$

$\Rightarrow m[5,5] + m[6,6] + P_4 \times P_5 \times P_6 = 0+0+1500 = 1500.$

$\boxed{m[5,6] = 1500;\ K=5.}$

$\Rightarrow m[6,7]; \Rightarrow K = i$ to $j-1 = 6$ to $6 = 6$.

$\Rightarrow m[6,6] + m[7,7] + P_5 \times P_6 \times P_7 = 0+0+30\times10\times5$
$= 1500$

$\boxed{m[6,7] = 1500;\ K=6.}$

for Diagonal (S=2)

⇒ m[1,3]; ⇒ K = i to j-1 = 1 to 2.

⇒ (K=1); ⇒ m[1,3] = m[1,1] + m[2,3] + $P_0 \times P_1 \times P_3$

= 0 + 2400 + 5x12x20 = 3600.

⇒ (K=2); ⇒ m[1,3] = m[1,2] + m[3,3] + $P_0 \times P_2 \times P_3$

= 600 + 0 + 5x10x20 = 1600

m[1,3] = 1600 ; K=2

⇒ m[2,4]; ⇒ K = i to j-1 = 2 to 3.

⇒ (K=2); ⇒ m[2,4] = m[2,2] + m[3,4] + $P_1 \times P_2 \times P_4$

= 0 + 1000 + 12x10x5 = 1600

⇒ (K=3); ⇒ m[2,4] = m[2,3] + m[4,4] + $P_1 \times P_3 \times P_4$

= 2400 + 0 + 12x20x5 = 3600.

m[2,4] = 1600 ; K = 2

⇒ m[3,5]; ⇒ K = i to j-1 = 3 to 4

⇒ (K=3); ⇒ m[3,5] = m[3,3] + m[4,5] + $P_2 \times P_3 \times P_5$

= 0 + 3000 + 10x20x30 = 9000

⇒ (K=4); ⇒ m[3,5] = m[3,4] + m[5,5] + $P_2 \times P_4 \times P_5$

= 1000 + 0 + 10x5x30 = 2500

m[3,5] = 2500; K=4

$\Rightarrow m[4,6]$; $\Rightarrow K = i$ to $j-1$; $\Rightarrow 4$ to 5 ; $(4,5)$

$\Rightarrow (K=4)$; $\Rightarrow m[4,6] = m[4,4] + m[5,6] + P_3 \times P_4 \times P_6$

$= 0 + 1500 + 20 \times 5 \times 10 = 2500.$

$\Rightarrow (K=5)$; $\Rightarrow m[4,6] = m[4,5] + m[6,6] + P_3 \times P_5 \times P_6$

$= 3000 + 0 * 20 \times 30 \times 10 = 9000.$

$\boxed{m[4,6] = 2500 ; K=4}$

$\Rightarrow m[5,7]$; $\Rightarrow K = i$ to $j-1$; 5 to 6.

$\Rightarrow (K=5)$; $\Rightarrow m[5,7] = m[5,5] + m[6,7] + P_4 \times P_5 \times P_7$

$= 0 + 1500 + 5 \times 30 \times 5 = 2250.$

$\Rightarrow (K=6)$; $\Rightarrow m[5,7] = m[5,6] + m[7,7] + P_4 \times P_6 \times P_7$

$= 1500 + 0 + 5 \times 10 \times 5 = 1750.$

$\boxed{m[5,7] = 1750 ; K=6}$

for Diagonal $(S=3)$

$\Rightarrow m[1,4]$; $\Rightarrow K = i$ to $j-1 = 1$ to 3.

$\Rightarrow (K=1)$; $\Rightarrow m[1,4] = m[1,1] + m[2,4] + P_0 \times P_1 \times P_4$

$= 0 + 1600 + 5 \times 12 \times 5 = 1900.$

$\Rightarrow (K=2)$; $\Rightarrow m[1,4] = m[1,2] + m[3,4] + P_0 \times P_2 \times P_4$

$= 600 + 1000 + 5 \times 10 \times 5 = 1850.$

$\Rightarrow (K=3)$; $\Rightarrow m[1,4] = m[1,3] + m[4,4] + P_0 \times P_3 \times P_4$

$= 1600 + 0 + 5 \times 20 \times 5 = 2100.$

$\boxed{m[1,4] = 1850 ; K=2}$

$\Rightarrow m[2,5]$; $\Rightarrow k = i$ to $j-1 = 2$ to 4.

$\Rightarrow (k=2)$; $\Rightarrow m[2,5] = m[2,2] + m[3,5] + P_1 \times P_2 \times P_5$

$= 0 + 2500 + 12 \times 10 \times 30 = 6100$.

$\Rightarrow (k=3)$; $\Rightarrow m[2,5] = m[2,3] + m[4,5] + P_1 \times P_3 \times P_5$

$= 2400 + 3000 + 12 \times 20 \times 30 = 12,600$.

$\Rightarrow (k=4)$; $\Rightarrow m[2,4] + m[5,5] + P_1 \times P_4 \times P_5$

$= 1600 + 0 + 12 \times 5 \times 30 = 3400$.

$$\boxed{m[2,5] = 3400;\ k=4.}$$

$\Rightarrow m[3,6]$; $\Rightarrow K = i$ to $j-1 = 3$ to 5.

$\Rightarrow (k=3)$; $\Rightarrow m[3,6] = m[3,3] + m[4,6] + P_2 \times P_3 \times P_6$

$= 0 + 2500 + 10 \times 20 \times 10 = 4500$.

$\Rightarrow (k=4)$; $\Rightarrow m[3,6] = m[3,4] + m[5,6] + P_2 \times P_4 \times P_6$

$= 1000 + 1500 + 10 \times 5 \times 10 = 3000$.

$\Rightarrow (k=5)$; $\Rightarrow m[3,6] = m[3,5] + m[6,6] + P_2 \times P_5 \times P_6$

$= 2500 + 0 + 10 \times 30 \times 10 = 5500$.

$$\boxed{m[3,6] = 3000;\ k=4}$$

$\Rightarrow m[4,7]$; $\Rightarrow K = i$ to $j-1 = 4$ to 6.

$\Rightarrow (k=4)$; $\Rightarrow m[4,7] = m[4,4] + m[5,7] + P_3 \times P_4 \times P_7$

$= 0 + 1750 + 20 \times 5 \times 5 = 2250$.

$\Rightarrow (k=5)$; $\Rightarrow m[4,7] = m[4,5] + m[6,7] + P_3 \times P_5 \times P_7$

$= 3000 + 1500 + 20 \times 30 \times 5 = 7500$

⇒ (K=6); ⇒ m[4,7] = m[4,6] + m[7,7] + $P_3 \times P_6 \times P_7$
= 2500 + 0 + 20 x 10 x 5 = 3500.

$$\boxed{m[4,7] = 2250; K=4}$$

For Diagonal (S=4)

⇒ m[1,5]; ⇒ K = i to j-1 = 1 to 4.

⇒ (K=1); ⇒ m[1,5] = m[1,1] + m[2,5] + $P_0 \times P_1 \times P_5$
= 0 + 3400 + 5 x 12 x 30 = 5200.

⇒ (K=2); ⇒ m[1,5] = m[1,2] + m[3,5] + $P_0 \times P_2 \times P_5$
= 600 + 2500 + 5 x 10 x 30 = 4600

⇒ (K=3); ⇒ m[1,5] = m[1,3] + m[4,5] + $P_0 \times P_3 \times P_5$
= 3600 + 3000 + 5 x 20 x 30 = 9600

⇒ (K=4); ⇒ m[1,5] = m[1,4] + m[5,5] + $P_0 \times P_4 \times P_5$
= 1850 + 0 + 5 x 5 x 30 = 2600.

$$\boxed{m[1,5] = 2600; K=4}$$

⇒ m[2,6]; ⇒ K = i to j-1; K = 2 to 5.

⇒ (K=2); ⇒ m[2,6] = m[2,2] + m[3,6] + $P_1 \times P_2 \times P_6$
= 0 + 3000 + 12 x 10 x 10 = 4200.

⇒ (K=3); ⇒ m[2,6] = m[2,3] + m[4,6] + $P_1 \times P_3 \times P_6$
= 2400 + 2500 + 12 x 20 x 10 = 7300.

⇒ (K=4); ⇒ m[2,6] = m[2,4] + m[5,6] + $P_1 \times P_4 \times P_6$.
= 1600 + 1500 + 12 x 5 x 10 = 3200.

⇒ (K=5); ⇒ m[2,6] = m[2,5] + m[6,6] + $P_1 \times P_5 \times P_6$
= 3400 + 0 + 12 x 30 x 10 = 7000.

$$\boxed{m[2,6] = 3200 ; K = 4}$$

$\Rightarrow m[3,7] ; \Rightarrow K = i\,\&\&\,j - 1 \Rightarrow K = 3\,\&\&\,6.$

$\Rightarrow (K=3) ; \Rightarrow m[3,7] = m[3,3] + m[4,7] + P_2 \times P_3 \times B$

$= 0 + 2250 + 10 \times 20 \times 5 = 3250$

$\Rightarrow (K=4) ; \Rightarrow m[3,7] = m[3,4] + m[5,7] + P_2 \times P_4 \times B$

$= 1000 + 1750 + 10 \times 5 \times 5 = 3000$

$\Rightarrow (K=5) ; \Rightarrow m[3,7] = m[3,5] + m[6,7] + P_2 \times P_5 \times P_7$

$= 2500 + 1500 + 10 \times 30 \times 5 = 5500$

$\Rightarrow (K=6) ; \Rightarrow m[3,7] = m[3,6] + m[7,7] + P_2 \times P_6 \times P_7$

$= 3000 + 0 + 10 \times 10 \times 5 = 3500.$

$$\boxed{m[3,7] = 3000 ; K = 4.}$$

for Diagonal ($S = 5$)

$\Rightarrow m[1,6] ; \Rightarrow K = i\,\&\&\,j - 1 = 1\,\&\&\,5.$

$\Rightarrow (K=1) ; \Rightarrow m[1,1] + m[2,6] + P_0 \times P_1 \times P_6$

$= 0 + 3700 + 5 \times 12 \times 10 = 4300.$

$\Rightarrow (K=2) ; \Rightarrow m[1,6] = m[1,2] + m[3,6] + P_0 \times P_2 \times P_6$

$= 600 + 3000 + 5 \times 10 \times 10 = 4100.$

$\Rightarrow (K=3) ; \Rightarrow m[1,6] = m[1,3] + m[4,6] + P_0 \times P_3 \times P_6$

$= 3600 + 2500 + 5 \times 20 \times 10 = 7100$

⇒ (K=4); ⇒ m[1,6] = m[1,4] + m[5,6] + P0 x P4 x P6.
= 1850 + 1500 + 5x5x10 = 3600.

⇒ (K=5); ⇒ m[1,6] = m[1,5] + m[6,6] + P0 x P5 x P6
= 2600 + 0 + 5x30x10 = 4100.

m[1,6] = 3600; K=4

⇒ m[2,7] ; ⇒ K = i to j-1 = 2 to 6.

⇒ (K=2); ⇒ m[2,7] = m[2,2] + m[3,7] + P1 x P2 x P7
= 0 + 3000 + 12x10x5 = 3600.

⇒ (K=3); ⇒ m[2,7] = m[2,3] + m[4,7] + P1 x P3 x P7
= 2400 + 2250 + 12x20x5 = 5850.

⇒ (K=4); ⇒ m[2,7] = m[2,4] + m[5,7] + P1 x P4 x P7
= 1600 + 1750 + 12x5x5 = 3650.

⇒ (K=5); ⇒ m[2,7] = m[2,5] + m[6,7] + P1 x P5 x P7
= 3400 + 1500 + 12x30x5 = 3700.

⇒ (K=6); ⇒ m[2,7] = m[2,6] + m[7,7] + P1 x P6 x P7
= 3700 + 0 + 12x13x5 = 4300

m[2,7] = 3600; K=2

for Diagonal (S=6)

⇒ m[1,7] ; ⇒ K = i to j-1 = 1 to 6.

⇒ (K=1); ⇒ m[1,7] = m[1,1] + m[2,7] + P0 x P1 x P7
= 0 + 3600 + 5 x 12 x 5 = 3900.

⇒ (K=2); ⇒ m[1,7] = m[1,2] + m[3,7] + P0 x P2 x P7
= 600 + 3000 + 5 x 10 x 5 = 3350.

⇒ (K=3); ⇒ m[1,7] = m[1,3] + m[4,7] + P0 x P3 x P7
= 3600 + 2250 + 5 x 20 x 5 = 6350.

⇒ (K=4); ⇒ m[1,7] = m[1,4] + m[5,7] + P0 x P4 x P7
= 1850 + 1750 + 5 x 5 x 5 = 3725.

⇒ (K=5); ⇒ m[1,7] = m[1,5] + m[6,7] + P0 x P5 x P7
= 2600 + 1500 + 5 x 30 x 5 = 4850

⇒ (K=6); ⇒ m[1,7] = m[1,6] + m[7,7] + P0 x P6 x P7
= 3600 + 0 + 5 x 10 x 5 = 3850

m[1,7] = 3725 ; K = 4

m[i,j] (i →

(j) ↓	1	2	3	4	5	6	7
7	3725	3600	3000	2250	1750	1500	0
6	3600	3700	3000	2500	1500	0	
5	2600	3400	2500	3000	0		
4	1850	1600	1000	0			
3	1600	2400	0				
2	600	0					
1	0						

A "longest Common Subsequence" (LCS)

"A" Subsequence is any Subset of an element of a sequence that maintain the same relative Order. If "A" is a Subsequence of "B", then it is denoted as $(A \subseteq B)$.

*) Characteristics (i) Recursive Sol^n.
(ii) Computing the length of LCS.

(i) "Recursive Sol^n" :- The recursive Sol^n to LCS problem involves establishing a recurrence for the value of an Optimal Sol^n.

⇒ Let $C[i,j]$ be the length of LCS of sequence x_i and y_j. If either $i=0$ & $j=0$, One of the sequence has length 0. So the LCS has length 0, The Optimal Substructure of the LCS problem given by the formula :-

$$C[i,j] = \begin{cases} 0; \text{ if } i=0, \& j=0 . \\ C[i-1, j-1]+1 ; \text{ if } j>0, x_i = y_j \\ \max(C[i-1, j-1], C[i-1, j]) \text{ if } i,j>0 \\ \qquad x_i \neq y_j \end{cases}$$

(ii) "Computing the length of LCS" :-

(i) Here two Sequence. $x_i = x_1, x_2 \cdots\cdots x_n$.
$y_j = y_1, y_2 \cdots\cdots y_n$ are passed as input to the procedure.

(ii) It stores the $c[i,j]$ value in a table i.e; $c[0, \cdots\cdots m, 0 \cdots\cdots n]$ whose entry are calculated in row major order.

(iii) It also maintains $B[1 \cdots\cdots m, 1 \cdots\cdots n]$ to simplify the construction of an Optimal Solution.

** "How to Create LCS table?

Q.1) Find LCS of $X = ABACA$ & $Y = BCBBA$.

(Ans):- Step → 1 No. of row in the table = m+1 (length X)
No. of Column in the table = n+1 (length of Y)

$y_j \longrightarrow$

x_i		B	C	B	B	A
	0	0	0	0	0	0
↓ A	0	0	0	0	0	1
B	0	1	1	1	1	1
A	0	1	1	1	1	2
C	0	1	2	2	2	2
A	0	1	2	2	2	3

LCS = BCA

Step → 2 fill the 1st row & 1st Column with 0.

Step → 3 Write x_i from row On wards & y_j from Column Onwards.

Step→4 Compute row $[x_i]$ with Column $[y_j]$.

(i) Case-I Equal $(x_i = y_j)$
Store [Diagonal Value + 1] and ↖.

(ii) Case-II Not equal $(x_i \neq y_j)$
find greater between top and left.

a.) If top > left.
then Store top value & ↑.

b.) If top == left → then Store top value and ↑.

c.) If top < left → then Store left value and ←.

Q 2.) x_i = PRESIDENT ; y_j = PROVIDENCE

(Ans):- No. of row in the table = m+1 = 10 (length of X)
No. of Columns in the table = n+1 = 10 (length of Y)

			P	R	O	V	I	D	E	N	C	E
		0	0	0	0	0	0	0	0	0	0	0
	P	0	1	←1	←1	←1	←1	←1	←1	←1	←1	←1
	R	0	1↑	2	←2	←2	←2	←2	←2	←2	←2	←2
x_i ↓	E	0	1↑	2↑	2↑	2↑	2↑	2↑	2	←3	←3	3↑
	S	0	1↑	2↑	2↑	2↑	2↑	2↑	3↑	3↑	3↑	3↑
	I	0	1↑	2↑	2↑	2↑	3	←3	3↑	3↑	3↑	3↑
	D	0	1↑	2↑	2↑	2↑	3↑	4	←4	←4	←4	←4
	E	0	1↑	2↑	2↑	2↑	3↑	4↑	5	←5	←5	5↑
	N	0	1↑	2↑	2↑	2↑	3↑	4↑	5↑	6	←6	←6
	T	0	1↑	2↑	2↑	2↑	3↑	4↑	5↑	6↑	6↑	6↑

y_j →

LCS = PRIDEN.

Date
Page

* "Algorithm" :-

LCS – length (X, Y)

1. m ← length [x]
2. n ← length [Y]
3. for i ← 1 to m.
4. do c [i, 0] ← 0.
5. for j ← 1 to n.
6. do c [0, j] ← 0.
7. for i ← 1 to m.
8. for j ← 1 to n.
9. do if $x_i = y_j$
10. then c [i, j] = c [i-1, j-1] + 1.
11. b [i, j] = "↑".
12. else if c [i-1, j] ≥ c [i, j-1].
13. then c [i, j] = c [i-1, j].
14. b [i, j] = "↑".
15. else c [i, j] = c [i, j-1].
16. b [i, j] = "←".
17. return c and b.

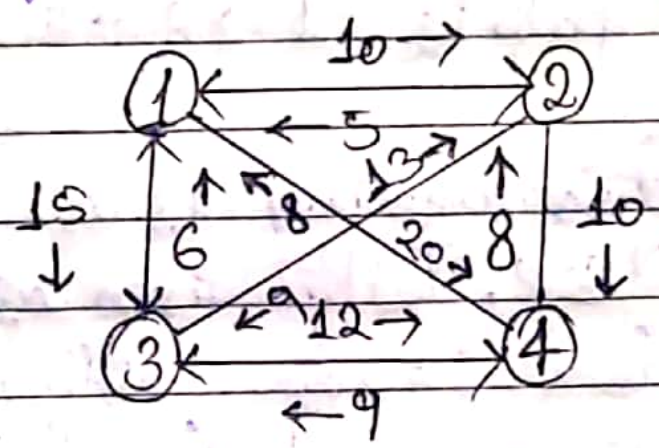
Running time Complexity = O(mn).

⭐ Travelling Sales Person Problem.

⇒ The traveling Salesman problems abide by a Salesman and a set of cities. The Salesman has to visit every one of the cities starting from a certain one (e.g; the hometown) and to return to the same City. The challange of the problem is that the traveling Salesman needs to minimize the total length of the trip.

⇒ Suppose the cities are $x_1, x_2 \ldots\ldots x_n$ where Cost C_{ij} denotes the cost of travelling from city x_i to x_j. The travelling salesperson problem is to find a route starting and ending at x_1 that will take in all cities with minimum Cost.

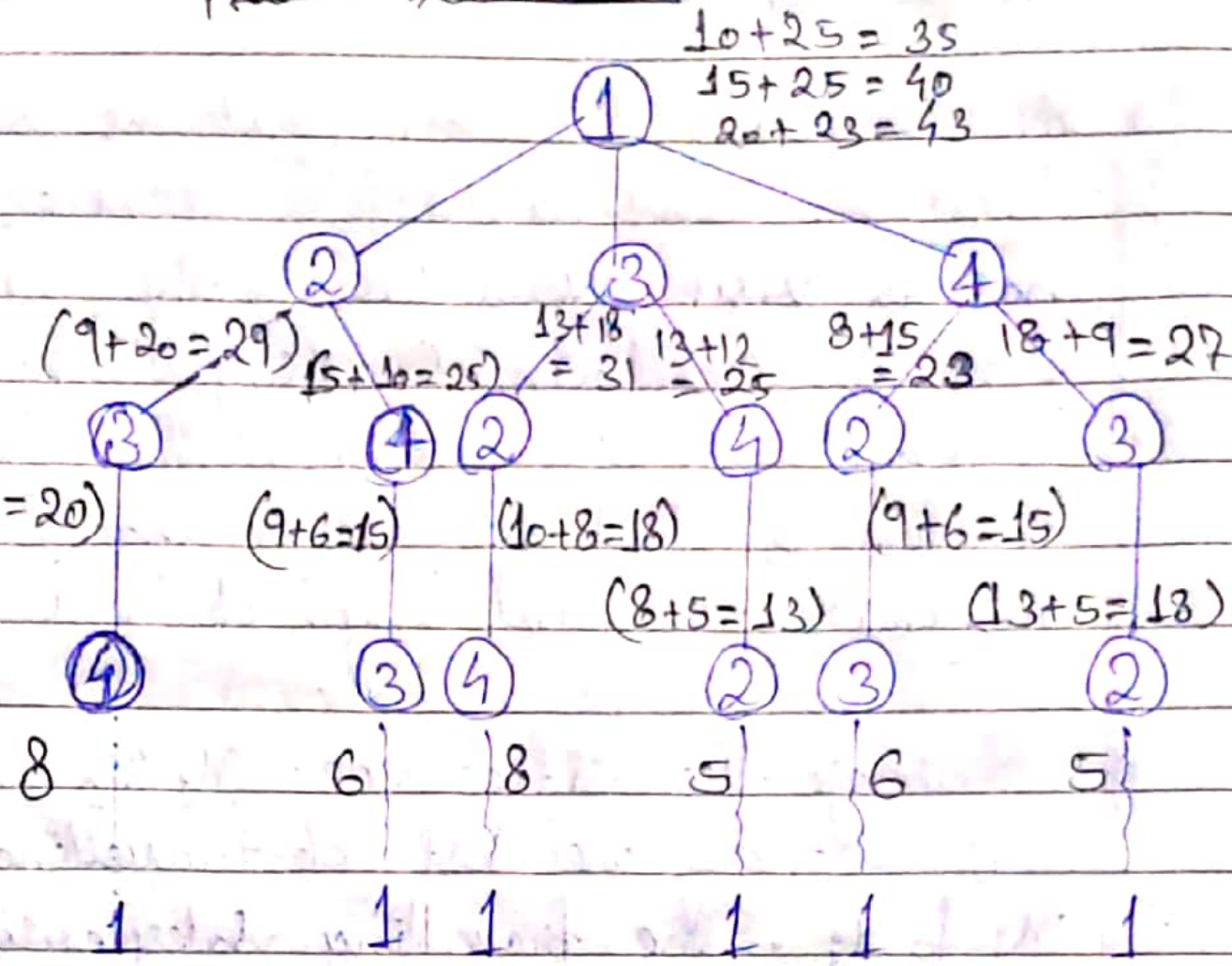
Q)



⇒ Cost - existency matrix.

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

⇒ "State Space tree":-



Cost of shortest root is = 35.

Shortest path is $1 \rightarrow 2 \rightarrow 4 \rightarrow 3$

(i) Dynamic Programming is a popular approach for solving TSP. TSP is given as, $G = [V, E]$ - where V = vertex; E = edge with weight - adjust [A].

(ii) Let us assume that (vertex 1) is starting Vertex of TSP tour.

(iii) The problem involve computation of minimum cost path i.e;- Starting from 1 & visit all other vertices exactly once.

(iv) Lets assume that, the cost of such path is Cost(i). The cost of TSP cycle = Cost(i) + Dist. of i
= Total TSP.

$C(i, \phi) = 1$ (Possibility node from 1) = 3 nodes.

$C(2, \phi) = C_{21} = 5$

$C(3, \phi) = C_{31} = 6$

$C(4, \phi) = C_{41} = 8$

(iii) → The general formula of TSP is,

i = Vertex taken $g(i,S) = \min_{k \in S} \{C_{ik} + g(k,\{S\}-k)\}$

S = Set of Vertex

Next possibility nodes = 6 nodes.

$g(2,\{3\}) = C_{23} + g(3,\phi) = C_{23} + 6 = 9+6 = 15.$

$g(2,\{4\}) = C_{24} + g(4,\phi) = 10+8 = 18.$

$g(3,\{2\}) = C_{32} + g(2,\phi) = 13+5 = 18.$

$g(3,\{4\}) = C_{34} + g(4,\phi) = 12+8 = 20.$

$g(4,\{3\}) = C_{43} + g(3,\phi) = 9+6 = 15.$

$g(4,\{2\}) = C_{42} + g(2,\phi) = 8+5 = 13.$

$g(2,\{3,4\}) = \min \begin{cases} C_{23} + g(3,\{4\}) \\ C_{24} + g(4,\{3\}) \end{cases} = \min \begin{cases} 9+20 \\ 10+15 \end{cases}$

$= \min \begin{cases} 29 \\ 25 \end{cases} = 25.$

$g(3,\{4,2\}) = \min \begin{cases} C_{34} + g(4,\{2\}) \\ C_{32} + g(2,\{4\}) \end{cases}$

$= \min \begin{cases} 12+13 \\ 13+18 \end{cases} = \min \begin{cases} 25 \\ 31 \end{cases} = 25.$

$g(4,\{2,3\}) = \min \begin{cases} C_{42} + g(2,\{3\}) \\ C_{43} + g(3,\{2\}) \end{cases} = \min \begin{cases} 23 \\ 27 \end{cases}$

$= 23.$

$$g(1,\{2,3,4\}) = \min \begin{cases} c_{12} + g(2,\{3,4\}) \\ c_{13} + g(3,\{2,4\}) \\ c_{14} + g(4,\{2,3\}) \end{cases}$$

$$= \min \begin{cases} 10+25 \\ 15+25 \\ 20+23 \end{cases} = \min 35.$$

"Algorithm" :-

1) Read the weight graph $G\langle V,E\rangle$

2) Initially, $d[i,j] = \begin{cases} \infty \text{ ; if } i,j \neq E(G) \\ 0 \text{ ; if } i=j \\ w_{ij} \text{ if edge } (i,j) \in E(G) \end{cases}$

3) Compute the function $g[i,S]$ that give the length of shortest path, starting from Vertex(i), travelling through all the vertices, in the set $\{S\}$.

Termination of Vertex(i) as follow.

$$C(i,\phi) = d[i,1] = 1.$$

4) Compute $g(i,S) = \min_{K \in S} \{ c_{ik} + g(K, S-\{K\}) \}$

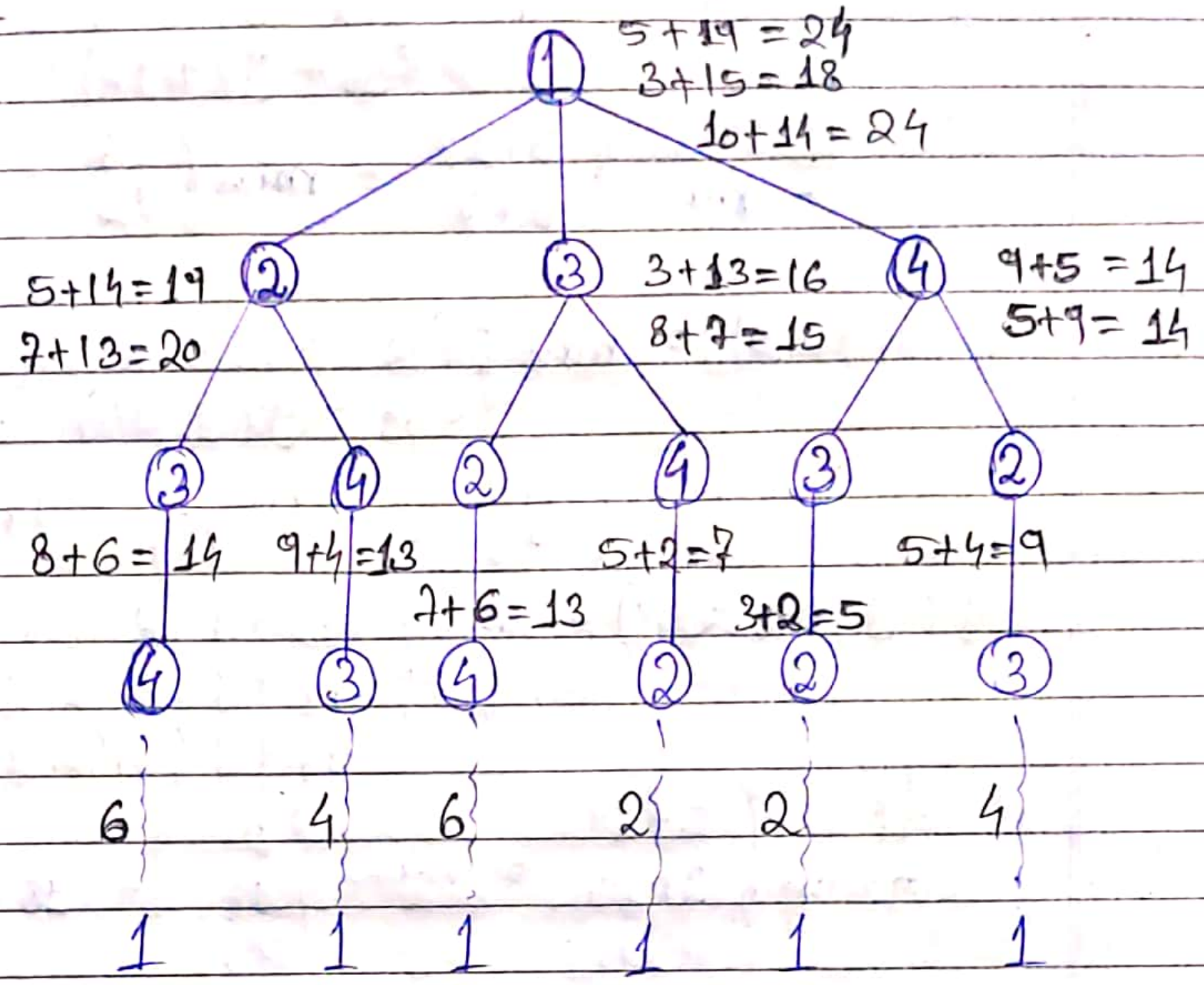
5) Return the value of $g(i,S)$

Q2) Solve

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

find the minimum weight of this matrix.

(Ans):-



$C(2,\phi) = 2$

$C(3,\phi) = 4$

$C(4,\phi) = 6.$

$g(2,\{3,4\}) = C_{23} + g(3,\phi) = C_{23} + 4 = 5+4 = 9.$

$g(2,\{4,1\}) = C_{24} + g(4,\phi) = 7+6 = 13.$

$g(3,\{2,1\}) = C_{32} + g(2,\phi) = 3+2 = 5.$

$g(3,\{4,1\}) = C_{34} + g(4,\phi) = 8+6 = 14.$

$g(4,\{2,1\}) = C_{42} + g(2,\phi) = 5+2 = 7.$

$g(4,\{3,1\}) = C_{43} + g(3,\phi) = 9+4 = 13.$

$$g(2,\{3,4\}) = \min\begin{cases} c_{23} + g(3,\{4\}) \\ c_{24} + g(4,\{3\}) \end{cases}$$

$$= \min\begin{cases} 5+14 \\ 7+13 \end{cases} = \min\begin{cases} 19 \\ 20 \end{cases} = 19.$$

$$g(3,\{2,4\}) = \min\begin{cases} c_{32} + g(2,\{4\}) \\ c_{34} + g(4,\{2\}) \end{cases}$$

$$= \min\begin{cases} 3+13 \\ 8+7 \end{cases} = \min\begin{cases} 16 \\ 15 \end{cases} = 15.$$

$$g(4,\{2,3\}) = \min\begin{cases} c_{42} + g(2,\{3\}) \\ c_{43} + g(3,\{2\}) \end{cases}$$

$$= \min\begin{cases} 5+9 \\ 9+5 \end{cases} = \min\begin{cases} 14 \\ 14 \end{cases} = 14.$$

$$g(1,\{2,3,4\}) = \min\begin{cases} c_{12} + g(2,\{3,4\}) \\ c_{13} + g(3,\{2,4\}) \\ c_{14} + g(4,\{2,3\}) \end{cases}$$

$$= \min\begin{cases} 5+19 \\ 3+15 \\ 10+14 \end{cases} = \min\begin{cases} 24 \\ 18 \\ 24 \end{cases} = 18.$$

So, Shortest Path = 1-3-4-2 = 18.

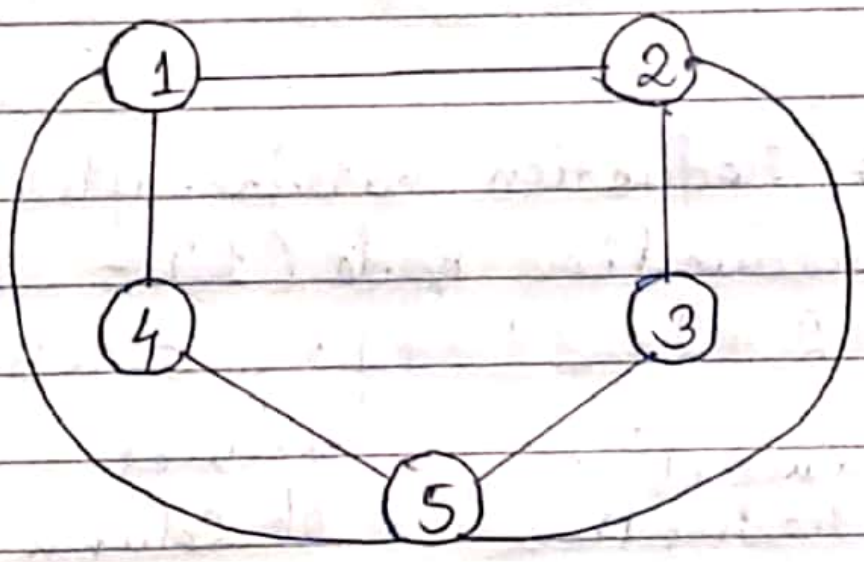
"Branch and Bound"

Branching

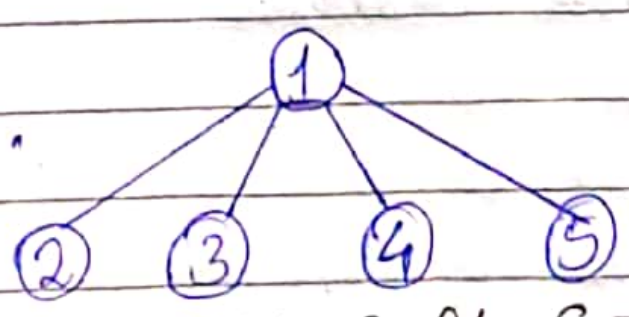
(i) This Branch and Bound involves decision of a given problem into 2 or more sub problem.

(ii) These sub problem are exclusively independently Problem with each other.
They are similar to Original problem but smaller in size and In this step union of all Visible Sol^n can be obtained.

Bounding It aids in limiting group of state-Space & in this step the best solution is identified.



	1	2	3	4	5
1	∞	20	30	10	11
2	15	∞	16	4	2
3	3	5	∞	2	4
4	19	6	18	∞	3
5	16	4	7	16	∞



C=35 C=53 C=24 C=31

Step-1 Go for row reduction.

	1	2	3	4	5	
1	∞	20	30	10	11	10
2	15	∞	16	4	2	2
3	3	5	∞	2	4	2
4	19	6	18	∞	3	3
5	16	4	7	16	∞	4

Subtract the lowest element from each entry of the matrix (Row Reduction)

	1	2	3	4	5
1	∞	10	20	0	1
2	13	∞	14	2	0
3	1	3	∞	0	2
4	16	3	15	∞	0
5	12	0	3	12	∞
	1	0	3	0	0

Step 2 Go for Column reduction :-

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	15	∞	0
5	11	0	0	12	∞

Step 3 Cost of reduction matrix after row and Column reduction node (1) =

$10 + 2 + 2 + 3 + 4 + 1 + 3 = 25$

(values of row reduction; values of Column reduction.)

final reduction Matrix.

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞

Step→4 The Path is gone from node 1 to 2. So, make 1st row & 2nd Column as ∞ Over the final reduction matrix.

∞	∞	∞	∞	∞
∞	∞	11	2	0
0	∞	∞	0	2
15	∞	12	∞	0
11	∞	0	12	∞

After row reduction. (As (2) to (1) there is no path. So, path of (2) to (1) will be Infinity)

	1	2	3	4	5	
1	∞	∞	∞	∞	∞	∞
2	∞	∞	11	2	0	0
≃ 3	0	∞	∞	0	2	0
4	15	∞	12	∞	0	0
5	11	∞	0	12	∞	0

Steps5 After row reduction the matrix will be same as every row as 0 as its lowest entry.

Step→6 After Column reduction.

So, the matrix will be same even after this because every Column has 0 as its lowest entry.

∞	∞	∞	∞	∞
∞	∞	11	2	0
0	∞	∞	0	2
15	∞	12	∞	0
11	∞	0	12	∞
		0	0	0

Cost of node (2) = c (1,2) + Reduction matrix Cost + r^n

= 10 + 25 + 0 + r^n = 35.

Step → 7 from Path (1) to (3). Make 1st row and 3rd Column = ∞

1	∞	∞	∞	∞
2	12	∞	∞	2 0
3	∞	3	∞	0 2
4	15	3	∞	∞ 0
5	11	0	∞	12 0

Step → 8 Again for row reduction.

∞	∞	∞	∞	∞
12	∞	∞	2	0
∞	3	∞	0	2
15	3	∞	∞	0
11	0	∞	12	0

Step → 9 Go for Column reduction

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	∞	2	0
3	∞	3	∞	0	2
4	15	3	∞	∞	0
5	11	0	∞	12	0
	11	0		0	0

	∞	∞	∞	∞	∞
	1	∞	∞	2	0
≃	∞	3	∞	0	2
	4	3	∞	∞	0
	0	0	∞	12	∞

Cost of node (3) = Cost (1, 3) + reduction Cost + rⁿ.
= 17 + 25 + 11 = 53.

Step :- 10

for Cost of (1,4) =

∞	∞	∞	∞	∞	0
12	∞	11	∞	0	0
0	3	∞	∞	2	0
∞	3	12	∞	0	0
11	0	0	∞	∞	0
0	0	0	∞		

Step :- 11 Go for row reduction & there will be not change entry.

Step :- 12 for Column reduction, there will be not change.

Cost of node (4) = cost(1,4) + reductioncost + R^n

= c(1,4) + 25 + 0.

= 25.

for node (1) to (5).

Row reduction =

∞	∞	∞	∞	∞	∞
12	∞	11	2	∞	2
0	3	∞	0	∞	0
15	3	12	∞	∞	3
∞	0	0	12	∞	0
0	0	0	0	∞	

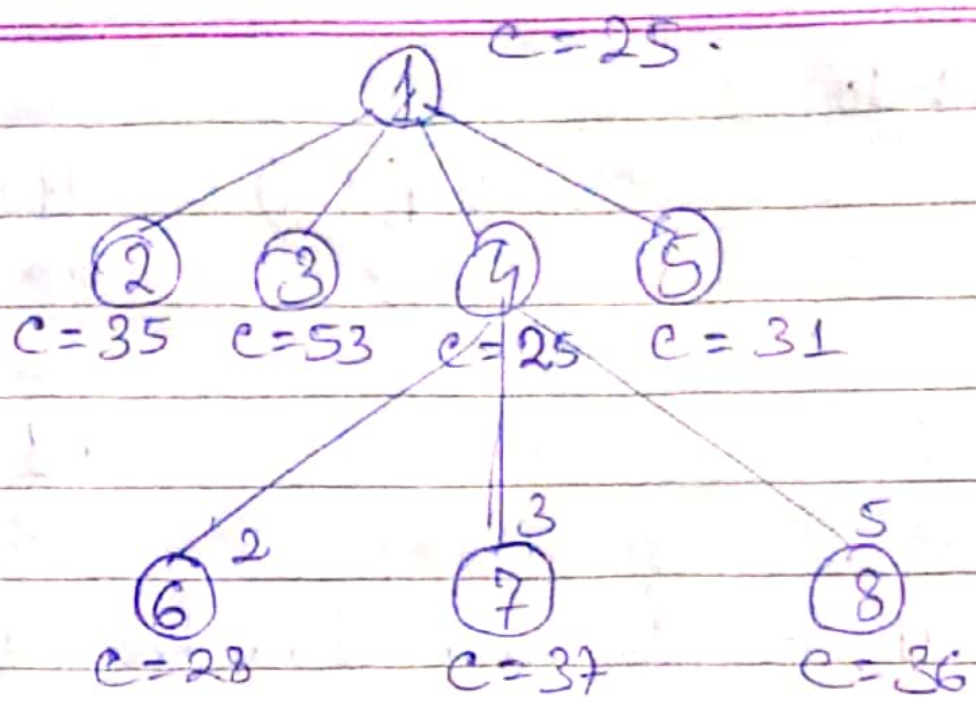
$\simeq$

∞	∞	∞	∞	∞
10	∞	9	0	∞
0	3	∞	0	∞
12	0	9	∞	∞
∞	0	0	12	∞
0	0	0	0	0

for Column reduction, there will be not change.

host of node (5) = Cost (1,5) + Reduction Cost + R^n

= 1 + 25 + 5 = 31.



Cost matrix of (4) =

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	3	∞	∞	2
4	∞	3	12	∞	0
5	11	0	0	∞	∞

for path 4 to 2 → make 4th row & 2nd column as infinity

	∞	∞	∞	∞	∞	∞
	12	∞	11	∞	0	0
=	0	∞	∞	∞	2	0
	∞	∞	∞	∞	∞	∞
	11	∞	0	∞	∞	0
	0	∞	0	∞	0	

Row reduction = no change,
Column reduction = no changes.

So, Cost of node (6) = Cost (4,2) + Cost reduction + r

= 3 + 25 + 0 = 28.

Now from (4 to (7) =

	∞	∞	∞	∞	∞	∞
	12	∞	∞	∞	0	0
	0	3	∞	∞	2	0
	∞	∞	∞	∞	∞	∞
	11	0	∞	∞	∞	0
	0	0	∞	∞	0	

Row reduction = Column reduction = no change.

Cost of (7) = Cost of (4,3) + reduction cost + r^n.

= 12 + 25 + 0 = 37.

From (4) to (8) =

∞	∞	∞	∞	∞
12	∞	11	∞	∞
0	3	∞	∞	∞
∞	∞	∞	∞	∞
11	0	0	∞	∞
0	0	0	∞	∞

∞ 11 0 ≃ 0 0 0

∞	∞	∞	∞	∞
1	∞	0	∞	∞
0	3	∞	∞	∞
∞	∞	∞	∞	∞
11	0	0	∞	∞
0	0	0	∞	0

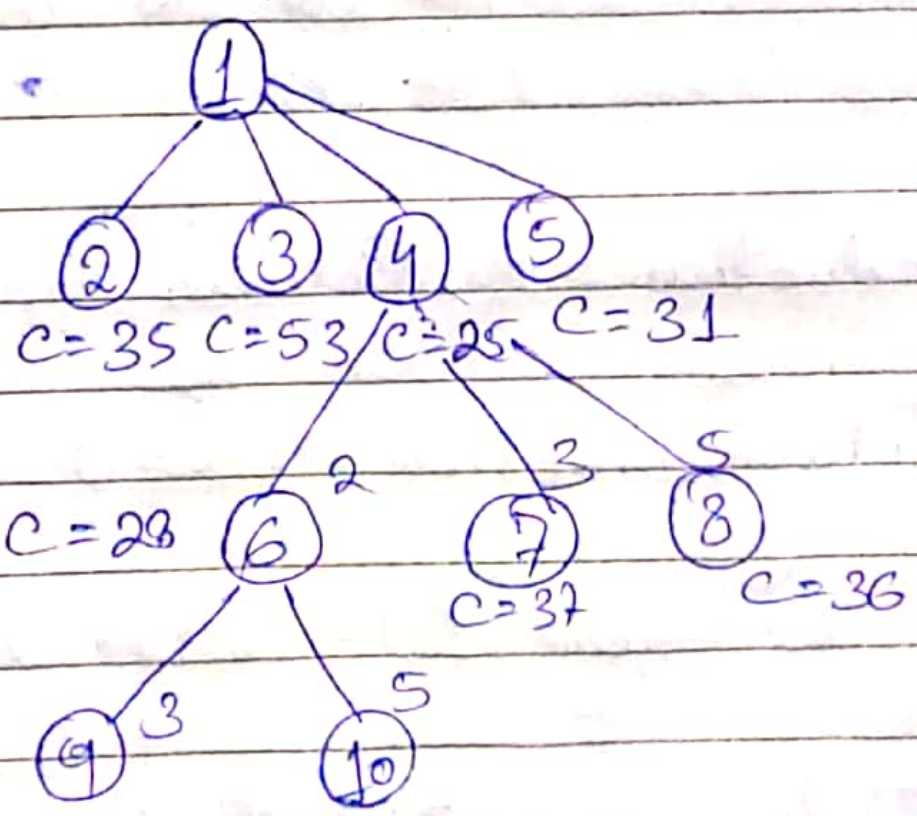
Row reduction = 11.

Column " = no change = 0.

Total reduction Cost = 11 + 0 = 11.

Cost of (8) = Cost of (4,5) + reduction + r^n

= 0 + 25 + 11 = 36.



Cost matrix of (6) =

∞	∞	∞	∞	∞
12	∞	11	∞	0
0	∞	∞	∞	2
∞	∞	∞	∞	∞
11	∞	0	∞	∞

for (2) to (3).

	∞	∞	∞	∞	∞	∞
	∞	∞	∞	∞	∞	∞
⇒	0	∞	∞	∞	2	0
	∞	∞	∞	∞	∞	∞
	11	∞	∞	∞	∞	11
					2	

$\simeq$

∞	∞	∞	∞	∞	
∞	∞	∞	∞	∞	
0	∞	∞	0	2	
15	∞	∞	∞	∞	
0	∞	∞	∞	∞	
0	∞	∞	0	2	

Row reduction = 11 , Column reduction = 2.
Total reduction Cost = 11+2 = 13.

Cost of (9) = Cost of (2,3) + reduction Cost + r^n.
= 11 + 28 + 13 = 52.

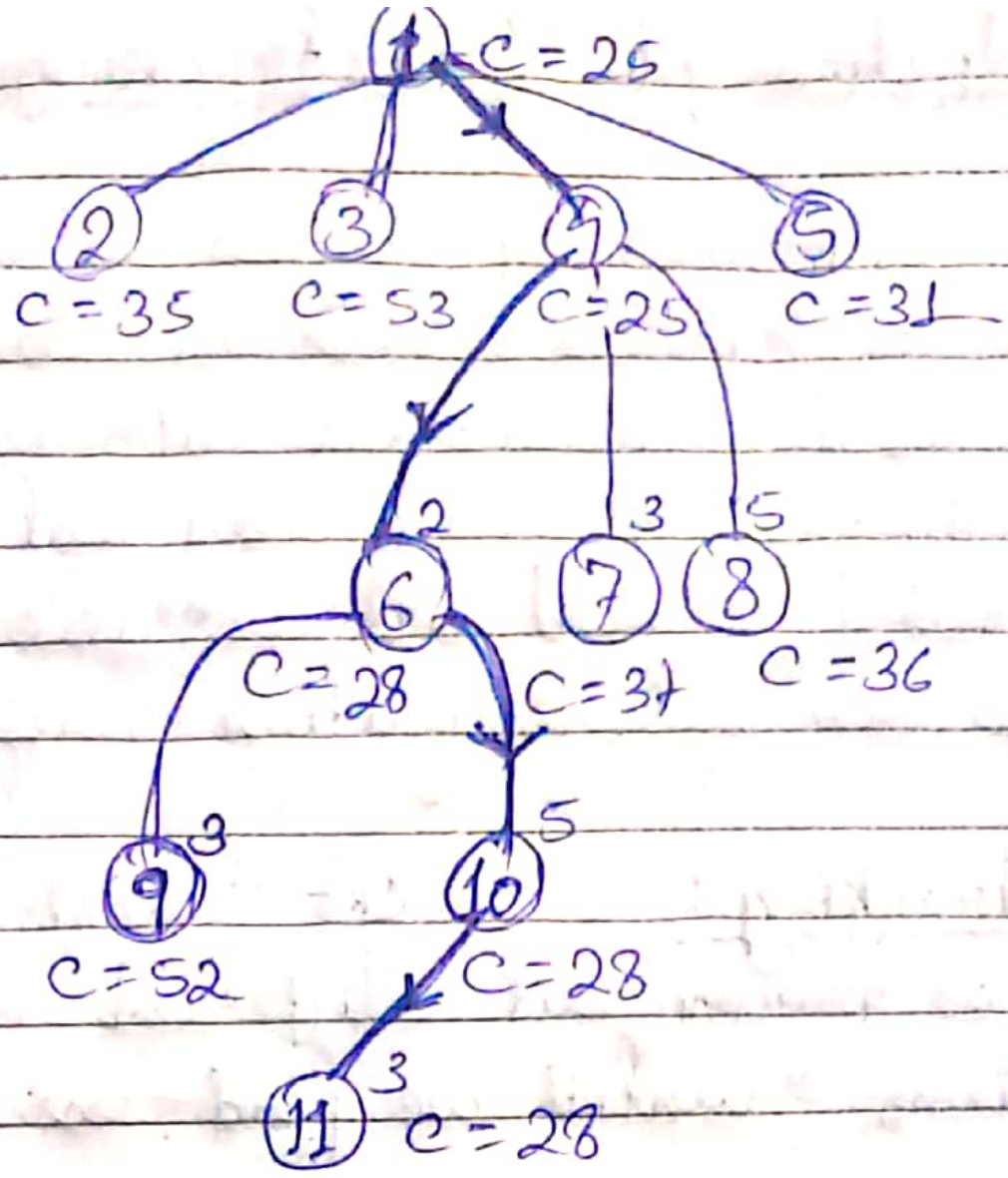
for (2) to (5) =

∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞
0	∞	∞	∞	∞	0
∞	∞	∞	∞	∞	∞
11	∞	0	∞	∞	0
0	∞	0	∞		

Row reduction = 0., Column reduction = 0.

Total reduction Cost = 0.

Cost of node (10) = Cost of (2,5) + reduction matrix + r^n
= 0 + 28 + 0 = 28.



from (5) to (3) =

∞	∞	∞	∞	∞	∞
∞	∞	∞	0	∞	∞
0	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞
∞	∞	0	∞	∞	0
0	∞	0	0	∞	

$\simeq$

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
0	∞	∞	∞	0
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞

Row reduction = Column reduction = 0.

Total Cost reduction = 0.

Cost of node (11) = Cost of (2,5) + reduction − Cost + x^n.

= 0 + 28 + 0 = 28.

So, the path is 1 - 4 - 2 - 5 - 3 - 1.

Total tour of Sales person is 10 + 6 + 2 + 7 + 3

= 28.

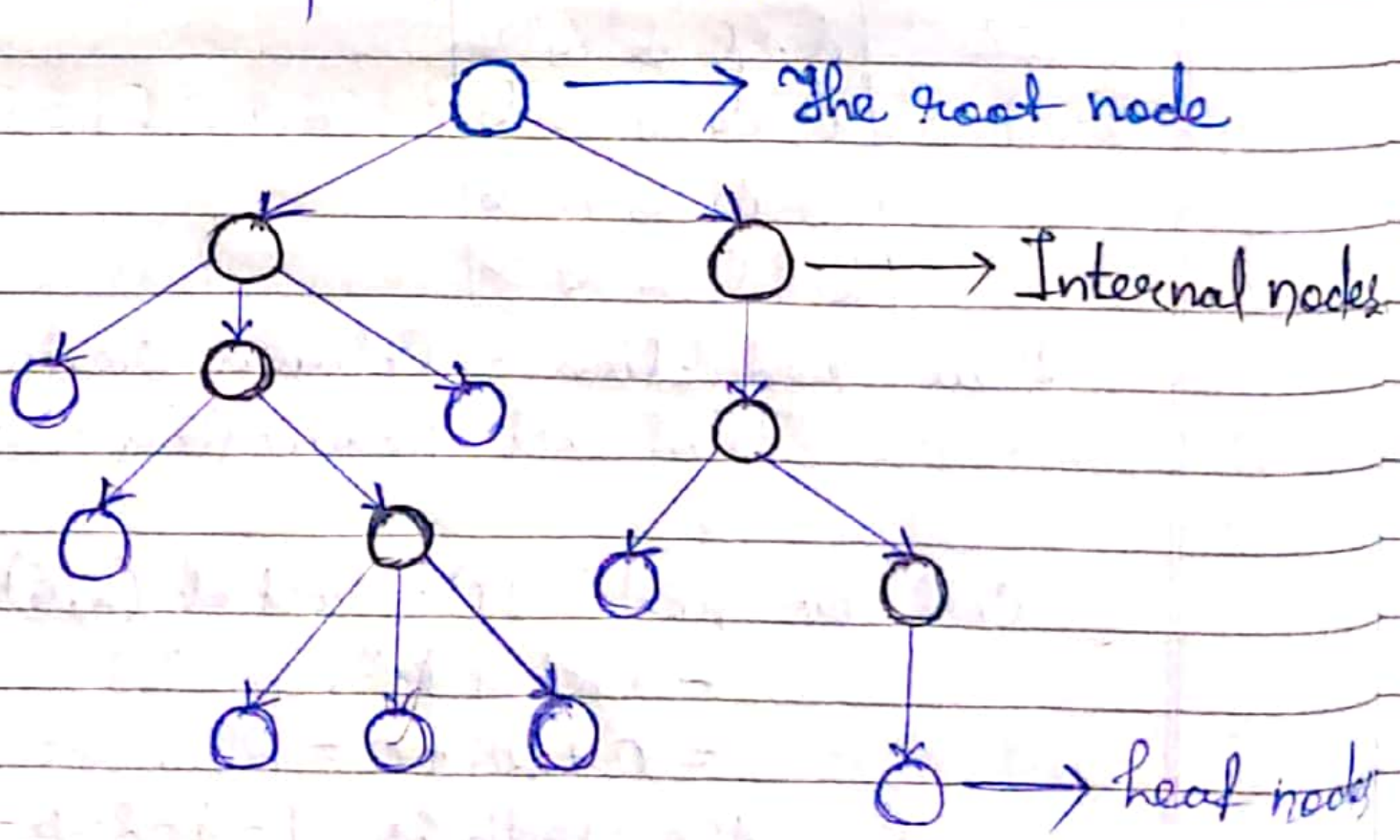
⊛ Introduction of Backtracking

The Backtracking is an algorithmic-method to solve a problem with an additional way. It uses a recursive approach to explain the problems. We can say that the backtracking is needed to find all possible combination to solve an Optimization problem.

⊛ **Backtracking :-** Back-tracking is a systematic way of trying out different sequences of decisions untill we find one that "works."

- Each non-leaf node in a tree is a parent of one (or) more other nodes (its children).
- Each node in the tree, other than the root, has exactly one parent.

** A tree is Composed of nodes.



⊛ Illustration of these techniques for problem-Solving

Backtracking algorithm determines the sol^n by systematically searching the sol^n space for the given problem.

⇒ Backtracking is a Depth-first search with any bounding function.

→ All solution using backtracking is needed to satisfy a complex set of constraints. The Constraints may be Explicit and Implicit.

⇒ Explicit Constraint is ruled, which restrict each vector element to be chosen from the given Set.

⇒ Implicit Constraint is ruled, which determine which each of the tuples in the sol^n space, actually satisfy the criterion function.

★ Bin Packing :- In the Bin Packing Problem, items of different volumes must be packed into a finite number of bins (or) containers each of a fixed given volume in a way that minimizes the number of bins used.

⇒ There are many Variations of this problem, such as 2D packing, Linear packing, Packing by weight, Packing by Cost and so on.

Applications :- (1) filling up Containers.
(2) loading trucks with weight Capacity constraints
(3) Creating file backups in media and technology

mapping in field - programmable gate array semiconductor chip design.

Bin Packing algorithms.

four thing you needs to know.

1) How to find the lower Bound for the problem.

2) How to perform the first-fit algorithm.

3) How to perform the first-fit dec. algorithm.

4) How to perform full-bin packing

* lower Bound

Ex-1 A plumber is using length of pipe 12 feet long and wishes to cut these length.

length	Number	
2	2	4
3	4	12
4	3	12
6	1	6
7	2	14
		48

=> 48 ÷ 12 = 4 pipes

* first-fit algorithm

A plumber is using lengths of pipe 12 feet long and wishes to cut these lengths.

2,2,3,3,3,3,1,1,1,6,7,7

length	Number
2	2
3	4
4	3
6	1
7	2

12: 2,2,3,3, wasted 2
12: 3,3,1, wasted 2
12: 4,4 wasted 4
12: 6 wasted 6
12: 7 wasted 5
12: 7 wasted 5

6 pipes = 24 ft. was wasted

First-fit decreasing algorithm.

A plumber is using length of pipe 12 feet long and wishes to cut these lengths.

2,2,3,3,3,3,4,4,4,6,7,7

lengths	Number
2	2
3	4
4	3
6	1
7	2

Arranging in a decreasing order
7,7,6,4,4,4,3,3,3,3,2,2

12: 7,4 waste 1
12: 7,4 waste 2
12: 6,4,2
12: 3,3,3,3
12: 2 waste 10

13 ft. wasted.

Full-bin packing.

A plumber is using length of pipe 12 feet long & wishes to cut these lengths.

2,2,3,3,3,3,4,4,4,6,7,7

Lengths	Number
2	2
3	4
4	3
6	1
7	2

⇒ 12: 7,3,2
⇒ 12: 7,3,2 — 4 pipes, 0 waste } optimal = (B
⇒ 12: 4,4,4.
⇒ 12: 3,3,6

ⓐ Knapsack TSP.

Q.) 0/1 Knapsack Problem.

m = 8 P = {1, 2, 5, 6}
n = 4 ; w = {2, 3, 4, 5}

⇒ x_1 = 0/1 x = {1, 0, 0, 0}
= {0, 0, 0, 0}
= {0, 0, 0, 1}
= {1, 1, 0, 0}.

It requires (2^n). No. of terms which is time Consuming so we follow Tabular method for time Consumption.

P	w
1	2
2	3
5	4
6	5

v \ w →	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8.

$V[i,w] = \max\{V[i-1,w], V[i-1, w-w(i)] + P[i]\}$

⇒ $V[4,4] = \max\{V[3,1], V[3,-4]+6\}$
=
⇒ $V[4,5] = \max\{V[3,5], V[3,5-5]+6\}$
$\max\{5, 0+6\} = 6$

$\Rightarrow V[4,6] = \max\{V[3,6], V[3,6-5]+6\}$
$= V[3,1]$

$= \max\{6, 0+6\} = 6$

$\Rightarrow V[4,7] = \max\{V[3,7], V[3,7-5]+6\}$

$= \max\{7, 1+6\} = 7.$

$\Rightarrow V[4,8] = \max\{V[3,8], V[3,8-5]+6\}$

$= \max\{7, 2+6\} = 8.$

$\{\underset{0}{x_1}\ \underset{1}{x_2}\ \underset{0}{x_3}\ \underset{1}{x_4}\}$ $\Rightarrow 8-6=2$

$2-2=0$

* Extra Method. (Set method) $\Rightarrow m=8 ; n=4$

$P=\{1,2,5,6\} ; w=\{2,3,4,5\}$

$\Rightarrow S^0 = \{(0,0)\}$

$\Rightarrow S_1^0 = \{(1,2)\}$

$\Rightarrow S^1 = \{(0,0),(1,2)\} \Rightarrow ; S_1^1 = \{(2,3),(3,5)\}$

$\Rightarrow S^2 = \{(0,0),(1,2),(2,3),(3,5)\}$

$\Rightarrow S_1^2 = \{(5,4),(6,6),(7,7),(8,9)\}$

$\Rightarrow S^3 = \{(0,0),(1,2),(2,3)(3,5),(5,4),(6,6),(7,7)\}$

$\Rightarrow S_1^3 = \{(6,5),(7,7),(8,8),(11,9),(12,11),(13,12)\}$

$\Rightarrow S^4 = \{(0,0), (1,2), (2,3), (5,4), (6,6), (6,5), (7,7), (8,8)\}$.

S^4

① $(8,8) \in S^4$

But $(8,8) \in S^3$ --- $n_4 = 1$.

$\Rightarrow (8-6, 8-5) = (2,3)$.

② $(2,3) \notin S^3$ ($\because n_3 = 0$).

and $(2,3) \in S^2$.

③ $(2,3) \in S^2$. ($\because S_2 = 1$).

but $(2,3) \notin S^1$

④ $(2-2, (3-3) = (0,0)$.

$(0,0) \in S^1$ and $(0,0) \in S^0$.

$\Rightarrow \{0,1,0,1\}$. (Ans).

⊛ "Huffman Coding"

(i) Data is stored in memory as bit.

(ii) 1 bit is either "0" or "1".

(iii) Coding means assign a code for the data.

(iv) Coding store data in specific format.

⊛ "Advantages of Coding":-

1⇒ **Data Compression :-** Coding decrease number of bit for the data. Hence less memory is required for the data.

2⇒ **Data Security :-** Coding provide data security using encryption technique.

Two types of Coding are there.

a⇒ Fix length Coding.
b⇒ Variable length Coding.

(a)⇒ **"Fix length Coding":-** Each code has fix no. of bit.

(b)⇒ **"Variable length Coding":-** Each code has variable length of bit. Huffman Coding is a variable length Coding.

⊛ "Steps of Huffman Coding" :-

Step→(1):- Arrange character in increasing order of freq.

Step→(2):- Create binary tree as follows.

left Child = 1st minimum frequency
Right Child = 2nd " "

Step→(3):- Parent node = left child + right child.

Step→(4):- Add Parent node to the list of elements.

Step→(5):- Repeat step 1, 2, 3, (n-1) times.

Step→(6):- Assign every left child as "0" and right child as "1".

Example→1

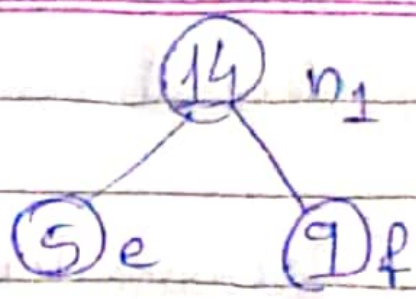
character	a	b	c	d	e	f
frequency	29	25	20	12	5	9

Sol^n→(i) Arrange frequency in Increasing Order.

e f d c b a
5 9 12 20 25 29

P.T.O.

(ii)

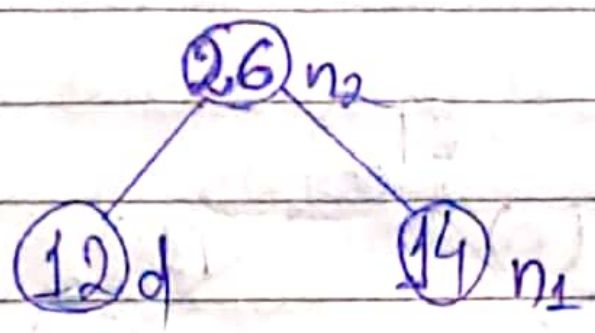


∴ left child = 5
Right child = 9
Parent child = 14.

(iii) Add "n_1" to next elements in list.

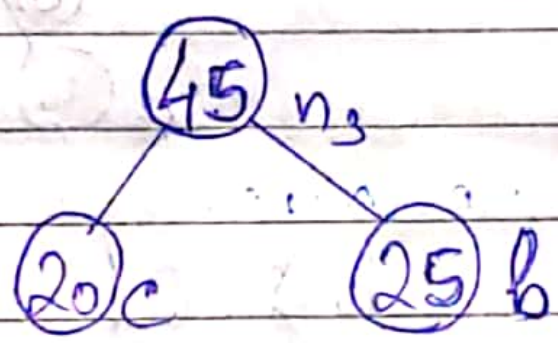
∴ d n_1 c b a (Arranged in Inc. Order)
12 14 20 25 29

∴



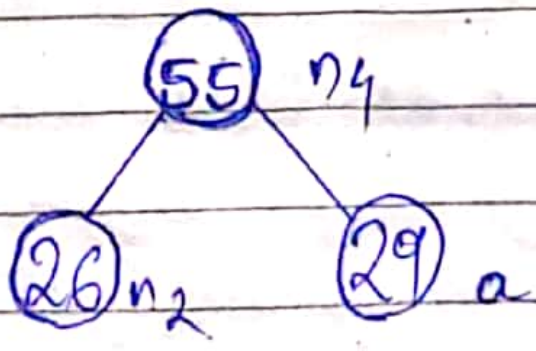
(iv) Add "n_2" to next elements in list.

c b n_2 a (Arranged in Inc. Order)
20 25 26 29



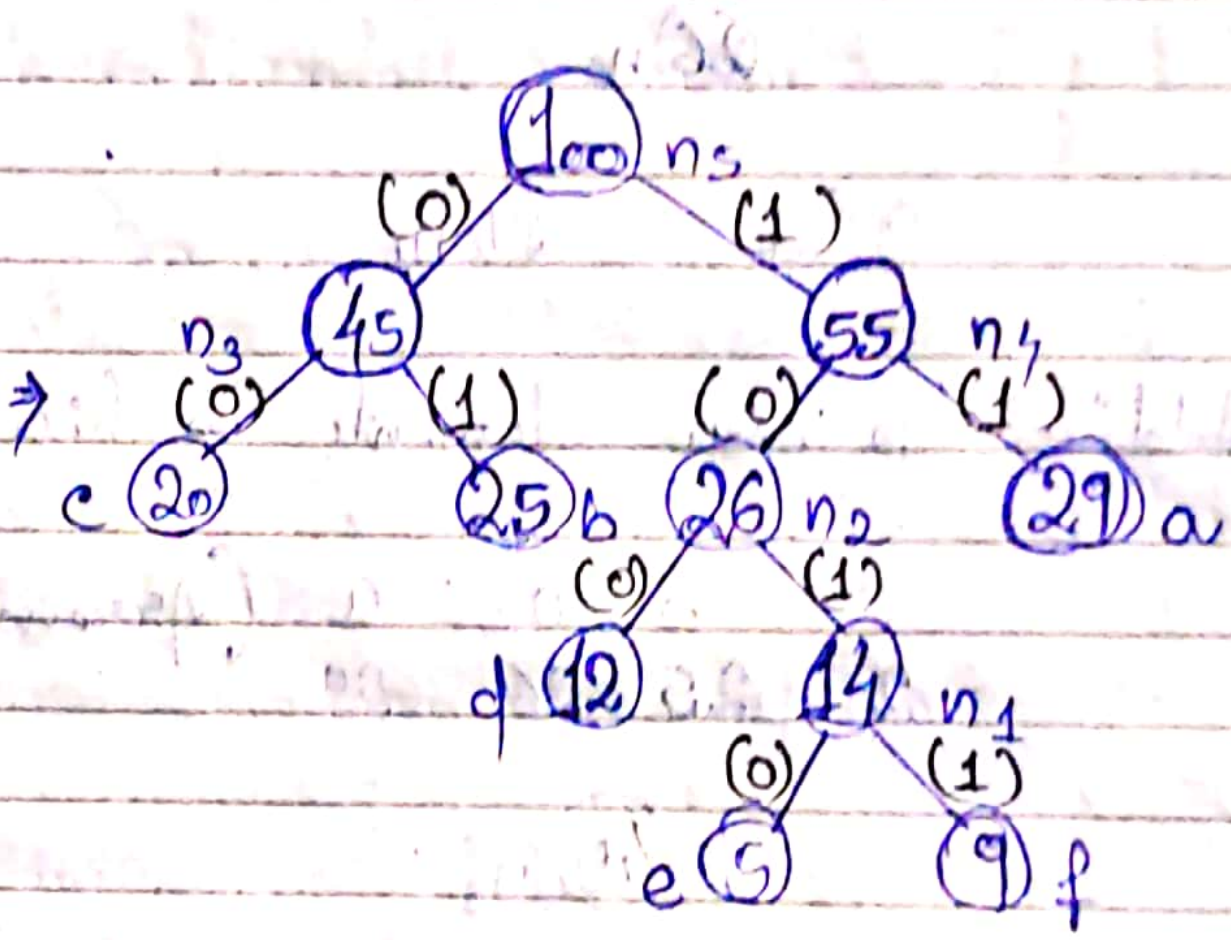
(v) Add "n_3" to next elements in list.

n_2 a n_3 (Arrange in Inc. Order)
26 29 45



(vi) Add "n_4" to next elements in the list.

n_3 n_4 (Arranged in Inc. Order)
45 55.



⇒ a = 11 ; b ⇒ 01 ; c ⇒ 00 ; d = 100
⇒ e ⇒ 1010 ; f = 1011.

"Fractional Knapsack Problem":-

Knapsack means bag, bag is to filled with different items and each item has profit and weight.

Weight or Capacity of the bag = W.

Ex:- Weight of the Knapsack = 15 Kg.
fill the Knapsack with item so that profit is max.

* Knapsack problem is of two type.

1.) fractional Knapsack.
2.) 0/1 Knapsack.

1.) fractional Knapsack.

a.) We can take fraction of item.

b.) It is a greedy algorithm.

2.) 0/1 Knapsack

(a) We Can't take fraction of a item.

(b) Dynamic Programming.

1) Fractional Knapsack Problem :-

(i) Arrange the item in decreasing order of profit by weight.

(ii) Put item one by one until the knapsack is full. If item taken = x_i, then Profit = $P_i x_i$ and weight = $w_i x_i$.

Mathematically, maximise $\sum_{i=1}^{n} P_i x_i$

Condition $\sum_{i=1}^{n} w_i x_i \leq W$.

* Ex:- $1\text{-}n = 3$; $W = 14$, $P_1, P_2, P_3 = 90, 100, 50$
$w_1, w_2, w_3 = 6, 8, 3$
Do the fractional Knapsack problem.

(Ans):-

Item	P_i (Profit)	w_i (weight)	P_i/w_i
Item 1	90	6	$90/6 = 15$
Item 2	100	8	$100/8 = 12.5$
Item 3	50	3	$50/3 = 16.6$

Arrange item in decreasing order of P_i/w_i
= item 3, item 1, item 2.

If $w_i \leq w \Rightarrow x_i = 1$, $w = w - w_i$
If $w_i \not\leq w \Rightarrow x_i = w/w_i \Rightarrow w = 0$

Item	$w_i \leq w$	x_i	$w = w - w_i$
Item 3	$3 \leq 14$	1	$w = 11$
Item 1	$6 \leq 11$	1	$w = 5$
Item 2	$8 \not\leq 5$	$x_i = \frac{w}{w_i} = 5/8$	0

(Q2) $n = 5, w = 9$; $P_1, P_2, P_3, P_4, P_5 = 10, 5, 6, 18, 3$
$\Rightarrow w_1, w_2, w_3, w_4, w_5 = 2, 3, 1, 4, 1$.

(Ans):-

Item	P_i (Profit)	w_i (weight)	P_i/w_i
Item 1	10	2	5
Item 2	5	3	1.6
Item 3	6	1	6
Item 4	18	4	4.5
Item 5	3	1	3

Item	$w_i \leq w$	x_i	w
Item 3	$1 \leq 9$	1	8
Item 1	$2 \leq 8$	1	6
Item 4	$4 \leq 6$	1	2
Item 5	$1 \leq 2$	1	1
Item 2	$3 \not\leq 1$	$x_i = w/w_i$ $= 1/3 = 0.3$	0

(*) "Algorithm"

```
Algorithm fractional Knapsack (P, W, n).
{
// P Store profit
   W Store weight.
   n Store fractional item
// Arrange item in decreasing Order ob profit.
   by weight.
```

for i ← 1 to n.
{
if $w_i \leq w$.
{
$x_i = 1$ // full item set is taken.

$w = w - w_i$.

else
{
$x_i = w/w_i$ // fractional item is taken.

$w = 0$
}
}

★ "Activity Selection procedure":—
In activity scheduling problem, we have to determine the Optimal no. of activity. Data schedule to be Resource.

Given a Set as $S = \langle 1, 2, 3, \ldots\ldots n \rangle$
i.e - "n" no. of activity is each Schedule to be used Some resource. where each Condition might arise, when multiple activity has a starting time (S_i) and finishing time (f_i). Each activity (A_i) has a starting time (S_i) and finishing time (f_i), where $0 \leq S_i \leq f_i \leq \infty$. for a given set of activity we can say that A_i & A_j are non-interfacing activity, if and Only if both have different starting and finishing time.

That makes their Start and finishing time don't Overlap $A_i(s_i, f_i)$ and $A_j(s_j, f_j)$.

So, for non-interfacing; $\boxed{A_i \cap A_j = \phi}$

"Procedure":-

Step→1 Select the 1st activity.

Step→2 Select the next activity whose $S_i \geq f_i$ of Previous selected activity.

Step→3 Repeat (Step→2) till all activity are checked.

Q1→

A_i	1	2	3	4	5	6
S_i	2	1	5	7	10	14
f_i	3	4	6	11	12	17

(Ans):- Select A_1, A_1 is is finished at 3.
But A_2 start at 1, So we can't take it.
A_3 start at 5, So A_3 is taken according to Condition $S_i \geq f_i$ and So On.
So, Maximum Compatible activity are (A_1, A_3, A_4, A_6).

Q2.)

A1	1	2	3	4	5	6	7	8	9	10	11
Si	1	3	0	5	3	5	6	8	8	2	12
fi	4	5	6	7	8	9	10	11	12	13	14

Consider the given activities with their given Si & fi. Compute a schedule in which largest no. of activity took place.

(Ans):- Man Compatible activity are = A_3, A_4, A_8, A_{11}.

** Some theory about Greedy And Dynamic Programming.

Q1.) what is Dynamic Programming?

(Ans):- Dynamic programming is an algorithm designing method that can be used when the solution to the sub-problem is view as a sequence of decision.

Q2.) what are the features of Dynamic Programming?

(Ans):- (i) Optimal Sol^n to the sub problem.

(ii) follows top-down approach.

(iii) Many no. of decision are generated.

(iv) Decision Sequence Containing Sub-Sequence that are Sub-optimal are not Considered.

(v). It definitely given an Optimal Sol^n.

Q3.) what are the draw backs of dynamic Programming?

(Ans):- (i) Time and Space requirement is high.

(ii) Optimality Should be checked at each level.

Q4) what are the general procedure of Dynamic Programming?

(Ans) (i):- Characterize the str. of an Optimal.

(ii):- Recursively define the value of an Optimal Sol^n.

(iii):- Compute the value of an Optimal Sol^n in top down fashion.

(iv):- Construct the Optimal Sol^n in a table from the Computed information.

Q5) what are the steps required to develop Greedy algorithm?

(Ans) (i):- Determine the Optimal Sub-str. of the Problem.

(ii):- Develop the recursive Sol^n :

(iii):- Prove that at any Stage of recursion.

(iv):- Develop a recursive Solution and Convert it into Iterative One.

Q6) what are the Similarity bet^n Greedy and Dynamic Programming?

(Ans) :- (i):- Both are used for Optimisation problem.

(ii):- Applied for Optimal Substructure method.

Dynamic Programming	Greedy Method.
1) Dynamic Programming is used to obtain the Optimal Sol".	1) Greedy Method is used to get the Optimal Sol".
2) In Dynamic Programming we choose at each step, but the choice may depend on the Sol" to Sub-problems.	2) In a greedy Algorithm, we make whatever choice seems best at the moment and then Solve the Sub-problems arising after the choice is made.
3) Less efficient as Compared to a greedy approach.	3) More efficient as Compared to a greedy approach.
4) Example : 0/1 Knapsack.	4) Example :- fractional Knapsack.
5) It is guaranteed that Dynamic Programming will generate an Optimal Sol" using Principle of Optimality.	5) In Greedy Method, there is no such guarantee of getting Optimal Sol".

Date
Page

⑧ Heuristic

↳ A Heuristic technique, is any approach to problem solving (or) self-discovery that employs a practical method that is not guaranteed to be optimal, perfect, or rotational, but which is nevertheless sufficient for reaching an immediate, short-term goal.

⇒ where finding an Optimal sol^n is impossible (or) impractical, heuristic methods can be used to speed up the process of finding a satisfactory sol^n.