Amit G. Mow

Page No.

Date / /

Chained Matrix Multiplication

→ We are given a chain of matrices $m_1, m_2, \ldots, m_n$ of n matrices to be multiplied that is to compute
$m = m_1 \cdot m_2 \cdot m_3 \cdots m_n$

⇒ eg: m_1, m_2 and m_3 have dimensions
$m_1 = 10 \times 100$, $m_2 = 100 \times 5$, $m_3 = 5 \times 50$ respectively.

→ To find out m_1, m_2, m_3 to be multiplied in two different ways using different paranthesization:

(I) $((m_1 \cdot m_2) \cdot m_3)$

This leads $10 \times 100 \times 5 = 5000$ Scalar Multiplications (S.M) to compute first $m_1 \cdot m_2$ and $10 \times 5 \times 50 = 2500$ S.M. to compute $[(m_1 \cdot m_2) \cdot m_3]$. Therefore total no. of 7500 scalar multiplications are used.

(II) $(m_1 \cdot (m_2 \cdot m_3))$

It Needs $100 \times 5 \times 50 = 25000$ S.M. for multiplying first $m_1 \cdot m_2$ and $10 \times 100 \times 50 = 50000$ S.M., so total of 75000 SM are required to multiply $(m_1 \cdot (m_2 \cdot m_3))$.

→ Therefore, 1st option (I) is 10 times faster then 2nd one.

⇒ eg. Similarly if we have m_1, m_2, m_3, m_4 then $m_1 \cdot m_2 \cdot m_3 \cdot m_4$ can be computed into 5 different ways.

(I) $((m_1 \cdot m_2)(m_3 \cdot m_4))$
(II) $((m_1 \cdot (m_2 \cdot m_3)) \cdot m_4)$
(III) $(m_1 \cdot ((m_2 \cdot m_3) \cdot m_4))$
(IV) $(((m_1 \cdot m_2) \cdot m_3) \cdot m_4)$
(V) $(m_1 \cdot (m_2 \cdot (m_3 \cdot m_4)))$

Problem Statement

→ The problem of Chained Matrix Multiplications is "Given a chain $m_1, m_2, \ldots m_n$ of n matrices where matrix m_i has dimensions $d_i \times d_i$ for $i = 1, 2, \ldots, n$. So, we have to fully paranthesize the product $m_1 \cdot m_2 \ldots m_n$ in such a way that minimizes the no. of scalar multiplications

→ Let $T(n)$ = no. of different ways to paranthesize a product of n matrices. If we make the 1st cut between i^{th} and $(i+1)^{st}$ matrices of the product then $m = (m_1 \cdot m_2 \ldots m_i) \cdot (m_{i+1} \ldots m_n)$

→ There are $T(i)$ ways to paranthesize the left hand term and $T(n-1)$ ways to paranthesize the right hand term.

→ So, $T(i)T(n-1)$ ways to paranthesize the whole term for $i = 1, 2, \ldots (n-1)$

Therefore $T(n) = \sum_{i=1}^{n-1} T(i)\, T(i-1)$ with $T(1) = 1$

n	1	2	3	4	5	10	15
$T(n)$	1	1	2	5	14	4862	2674440

→ These $T(n)$ values are called "Catalan Numbers".

→ Finding the best way to calculate M using the direct approach requires $\Omega(4^n/n)$ time, which is impractical for large values of n.

→ But Principle of Optimality applies to this problem as follows:

→ If the optimal way to multiply all matrices $m_1, m_2, \ldots, m_n$ requires to make the first cut between i^{th} and $(i+1)^{th}$ matrices of the product

the both the subproducts $m_1, m_2 \ldots m_i$ and $m_{i+1} \cdot m_{i+2} \ldots m_n$ must be also calculated into an optimal way.

→ For this, we construct a table $m[1 \ldots n, 1 \ldots n]$ where m_{ij} gives the optimal solution for the part $m_i, m_{i+1} \ldots m_j$, $i = 1, 2, \ldots, n$ & $j = 1, 2 \ldots n$.

→ We construct the table m diagonal by diagonal as follows:

→ Diagonal S contains the elements m_{ij} such that $j - i = s$

(I) Diagonal $s = 0$ contains the elements m_{ij}, $i = 1, 2, \ldots n$, which is $m_{ij} = 0$.

(II) Diagonal $s = 1$ contains the element $m_{i,i+1}$ where $m_i m_{i+1}$ using $d_{i-1} \times d_i \times d_{i+1}$ scalar multiplications

(III) when $s > 1$, the diagonal s contains $m_{i,i+s}$ corresponding to $m_i \cdot m_{i+1} \ldots m_{i+s}$, so, we fill the table m_{ij} using $s = 0, 1, \ldots n$

$S = 0$: $m_{ii} = 0$, $i = 1, 2 \ldots n$

$S = 1$: $m_{i,i+1} = d_{i-1} \cdot d_i \cdot d_{i+1}$, $i = 1, 2, \ldots, n-1$

$S = 2 \ldots (n-1)$:

$$m_{i,i+s} = \min_{k = i \ldots i+s-1} \left[m_{ik} + m_{k+1, i+s} + d_{i-1} \cdot d_k \cdot d_{i+s} \right]$$

$, i = 1, 2, \ldots, n-s.$

Ex: Problem 1 Perform Matrix Multiplication given $A = 13 \times 5$, $B = 5 \times 89$, $C = 89 \times 3$ and $D = 3 \times 34$ using chain matrix multiplication to obtain answer with minimum number of multiplications.

m_{ij} ($i \to$, $j \downarrow$)	1	2	3	4	
1	0	$13 \times 5 \times 89$ = 5785	k=1 1530	k=3 2856	S=3
2		0	$5 \times 89 \times 3$ k=2 1335	k=3 1845	S=2
3			0	$89 \times 3 \times 34$ 9078	S=1
4				0	S=0

Array.	0	1	2	3	4
D	13	5	89	3	34

$\Rightarrow$ Initially $S = 0$, we have $m_{11} = 0$, $m_{22} = 0$, $m_{33} = 0$ as $m_{ii} = 0$ (for $i = 1 \ldots n$)

$\Rightarrow$ $S = 1$, we have $i = 1, 2, 3$, as $i = 1, \ldots (n-1)$

① $i = 1$

$m_{12} = \underset{d_{i-1}}{13} \times \underset{d_i}{5} \times \underset{d_{i+1}}{89} = 5785$

② $i = 2$

$m_{23} = 5 \times 89 \times 3 = 1335$

③ $i = 3$

$m_{34} = 89 \times 3 \times 34 = 9078$

$\Rightarrow$ $S = 2$, we have $i = 1, 2$ as $[i = 1, 2, \ldots (n-s)]$

① $i = 1$, $k = i \ldots i + s - 1$

$\Rightarrow k = 1 \ldots 1 + 2 - 1$

$\Rightarrow k = 1 \ldots 2$

$\Rightarrow$ $i = 1, k = 1, S = 2$

$m_{i, i+s} \Rightarrow m_{13} = m_{11} + m_{23} + d_0 \cdot d_1 \cdot d_3$

$= \cancel{5785}\,0 + 1335 + (13 \times 5 \times 3) = 1530$

DOMS Page No.
Date / /

$\Rightarrow i=1,\ k=2,\ s=2$

$m_{i,i+s} \Rightarrow m_{13} = m_{12} + m_{33} + d_0 \cdot d_2 \cdot d_3$

$= 5785 + 9078 + (13 \times 89 \times 3)$

$= 18834$

Now, min $\Rightarrow m_{13} \Rightarrow \min(1530, 18334)$

$m_{13} \boxed{= 1530} \longrightarrow k = 1$

(2) $i=2$, we have $k = i \ldots i+s-1$

$= 2 \ldots 2+2-1$

$k = 2, 3$

$\rightarrow$ $s=2,\ i=2,\ k=2$

$\therefore m_{i,i+s} \Rightarrow m_{24} = m_{22} + m_{34} + d_1 \cdot d_2 \cdot d_4$

$= 0 + 9078 + (5 \times 89 \times 34)$

$= 24208$

$\Rightarrow s=2,\ i=2,\ k=3$

$\therefore m_{i,i+s} \Rightarrow m_{24} = m_{23} + m_{44} + d_1 \cdot d_3 \cdot d_4$

$= 1335 + 0 + (5 \times 3 \times 34)$

$= 1845$

Now min $\Rightarrow m_{24} = \min(24208, 1845)$

$\boxed{m_{24} = 1845} \longrightarrow k = 3$

Date / /

$\Rightarrow$ $\underline{S=3}$, we have $i = 1$

$\cdot$ $p=1, s=3, k = i \ldots p+s-1$

$\Rightarrow k = 1 \ldots 1+3-1$

$\Rightarrow k = 1, 2, 3$

$\therefore \Rightarrow s=3, p=1, k=1$

$\therefore m_{i,i+s} \Rightarrow m_{14} = m_{11} + m_{24} + d_0 \cdot d_1 \cdot d_4$

$= 0 + 1845 + (13 \times 5 \times 34)$

$= 4055$

$\Rightarrow s=3, i=1, k=2$

$m_{i,i+s} \Rightarrow m_{14} = m_{12} + m_{34} + d_0 \cdot d_2 \cdot d_4$

$= 5785 + 9078 + (13 \times 89 \times 34)$

$= 54201$

$\Rightarrow s=3, i=1, k=3$

$m_{i,i+s} \Rightarrow m_{14} = m_{13} + m_{44} + d_0 \cdot d_3 \cdot d_4$

$= 1530 + 0 + (13 \times 3 \times 34)$

$= 2856$

So, $\min \Rightarrow m_{14} = \min(4055, 54201, 2856)$

$= 2856$

$\therefore$ $\boxed{2856}$ no. of minimum scalar multiplications are required.

$\rightarrow$ This table just gives the no. of scalar multiplications, so, if we want to keep track of choices made, we use keep best 'k', when we compute m_{ij}, we save best $k[i,j] = k$.

→ Here k is the value corresponding to the minimal term. Best $k[l,n]$ tells us where to make first cut in the product.

→ Proceeding recursively on both that terms $m_i \ldots m_k$ & $m_{k+1} \ldots m_j$, we construct an optimal way of parathesized M_i.

⇒ Parantheis Calculations.

$k=0 \quad k=1 \quad k=2 \quad k=3 \quad k=4$

$| A | B | C | D |$

Now for $s=2$, we have $k=1$ and $k=3$

$\therefore$ $k=1 \quad k=3$

$A \mid B \; C \mid D \Rightarrow A\,(BC)\,D$

$\therefore$ B and C are first multiplied $\Rightarrow 5 \times 89 \times 3 = 1335$ S.M.

⇒ Now for Next $s=3$, we have $k=3$

$\therefore$ $k=3$

$A \; B \; C \mid D \Rightarrow (A\,(BC))\cdot D$

$\Rightarrow (A(BC))$

$\Rightarrow A = 13 \times 5$ and $BC = 5 \times 89 \times 3$

$\Rightarrow A(BC) = 13 \times 5 \times 3$

$= 195$

and Next $(A(BC))\cdot D \Rightarrow$

$A\cdot(BC) = 13 \times 5 \times 3$ & $D = 3 \times 34$

$\therefore (A(BC))D = 13 \times 3 \times 34$

$= 1326$

$(A\cdot(BC))\cdot D = 1530 + 195 + 1326$

$= 2856$ S.M.

Ans - $((A(BC)) \cdot D)$ with 2856 S.M. operations

Ex2 Consider the chain of $A_1, A_2 \ldots, A_6$ with dimensions given below. Give the optimal parenthesization to get the product ~~$A_1 \cdot A_6$~~ $A_2 \to A_5$

× A1 :	30 × 35
A2 :	35 × 15
A3 :	15 × 5
A4 :	5 × 10
A5 :	10 × 20
× A6 :	20 × 30

i \ j	1	2	3	4	
1	0	(35×15×5) 2625	(k=2) 4375	(k=2) 7125	s=3
2		0	(15×5×10) 750	(k=2) 2500	s=2
3			0	(5×10×20) 1000	s=1
4				0	s=0

0	1	2	3	4
35	15	5	10	20

Now Initially,

⇒ For $s=0$, we have all $M_{11}, m_{22}, M_{33}, m_{44} = 0$

⇒ For $s=1$, we have $i = 1, 2, 3$.

$\therefore m_{12} = 35 \times 15 \times 5 = 2625$

$m_{23} = 15 \times 5 \times 10 = 750$

$m_{34} = 5 \times 10 \times 20 = 1000$

⇒ For $s=2$, we have $i = 1, 2$

(I) For $s=2$, $i=1$, we have $k = 1, 2$

(1) $s=2$, $i=1$, $k=2$

$m_{13} = m_{11} + m_{23} + (35 \times 15 \times 10)$

$= 750 + 2500$

$= 6000$

(2) $s=2,\ i=1,\ k=2$

$m_{13} = m_{12} + m_{33} + d_0 d_2 d_3$

$= 2625 + 0 + (35 \times 3 \times 10)$

$\boxed{= 4375 \rightarrow \text{Min}}$

(II) $s=2,\ i=2$, we have $k=2,3$

(1) $s=2,\ i=2,\ k=2$

$m_{24} = m_{22} + m_{34} + d_1 d_2 d_4$

$= 0 + 1000 + (15 \times 5 \times 20)$

$\boxed{= 2500 \rightarrow \text{Min}}$

(2) $s=2,\ i=2,\ k=3$

$m_{24} = m_{23} + m_{44} + d_1 \cdot d_3 \cdot d_4$

$= 750 + 0 + (15 \times 10 \times 20)$

$= 3750$

$\Rightarrow$ For $s=3$, we have $i=1$

For $s=3,\ i=1$, we have $k=1,2,3$

(1) $s=3,\ i=1,\ k=1$

$m_{14} = m_{11} + m_{24} + d_0 d_1 d_4$

$= 0 + 2500 + (35 \times 15 \times 20)$

$= 13000$

(2) $s=3,\ i=1,\ k=2$

$m_{14} = m_{12} + m_{34} + d_0 d_2 d_4$

$= 2625 + 1000 + 35 \times 5 \times 20$

$\boxed{= 7125 \rightarrow \text{Min}}$

(3) $s=3,\ i=1,\ k=3$

$m_{14} = m_{13} + m_{44} + d_0 d_3 d_4$

$= 4375 + 0 + 35 \times 10 \times 20$

$= 11375$

$\Rightarrow$ Now Paranthesization

$k=2 \quad k=2 \longrightarrow s=2$

$k=1$

$A \mid \cancel{B} \ \cancel{C} \ \cancel{D} \quad A \quad B \mid\mid C \quad D$

$\therefore (A \ B)\ (C \cdot D)$

Now $(A \cdot B) \cdot \mid (C \cdot D)$

$k=2 \rightarrow s=3$

$\boxed{\therefore ((A \cdot B) \cdot (C \cdot D))}$

Longest Common Subsequence

Subsequence: A subsequence of a given sequence is a sequence with 0 or more elements left out from the given sequence.

→ Given a sequence $X = \langle x_1, x_2, x_3, \ldots x_n \rangle$ & another sequence $Z = \langle z_1, z_2, z_3, \ldots, z_k \rangle$ is a subsequence of X, if there exists a strictly increasing sequence $\langle i_1, i_2, \ldots, i_k \rangle$ of indices of x such that for all $j = 1, 2, \ldots k$, we have $x_{i_j} = z_j$

eg: $X = \langle A, B, C, B, D, A, B \rangle$
$Z = \langle B, C, D, B \rangle$

⇒ **Common Subsequence**

Given two sequences of X & Y, we say that a sequence Z is a common subsequence of X & Y, if Z is a common subsequence of both X & Y.

e.g. $X = \langle a, b, c, b, d, a, b \rangle$ &
$Y = \langle b, d, c, a, b, a \rangle$

The sequence $\langle b, c, a \rangle$ of length '3' is a common subsequence of both X & Y. Subsequence $\langle b, c, b, a \rangle$ of length '4' is also a common subsequence of X & Y.

→ The sequence b, c, b, a is a longest common subsequence (LCS) of X & Y. because there is no common sequence of length 5 or greater.

Problem Statement:
→ Longest Common Subsequence Problem. (LCS Problem)

We are given 2 sequences $X = \langle x_1, x_2, x_3, \ldots x_m \rangle$ & $Y = \langle Y_1, Y_2, Y_3, \ldots, Y_n \rangle$, we have to find a maximum length common subsequence of X & Y.

→ Prefix of sequence:-
Given a sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$ we define the ith prefix of X as $X_i = \langle x_1, x_2, x_3, \ldots, x_i \rangle$, where $i = 0, 1, 2, \ldots, m$.
eg: if $X = \langle A, B, C, B, D, A, B \rangle$, then $X_4 = \langle A, B, C, B \rangle$ and $X_0 = \langle \rangle$.

* Solving LCS using Dynamic Programming.
(1) Principle of Optimality:-

LCS has an optimal substructure property.
Theorem:
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ & $Y = \langle Y_1, Y_2, \ldots, Y_n \rangle$ be sequence & let $Z = \langle Z_1, Z_2, \ldots, Z_k \rangle$ be any common LCS of X & Y.

(I) If $X_m = Y_n$, then $Z_k = X_m = Y_n$ & Z_{k-1} is a LCS of X_{m-1} & Y_{n-1}.
(II) If $X_m \neq Y_n$, then $Z_k \neq X_m \Rightarrow Z$ is an LCS of X_{m-1} & Y.
(III) If $X_m \neq Y_n$, then $Z_k \neq Y_n \Rightarrow Z$ is an LCS of X & Y_{n-1}.

This theorem shows that an LCS of two sequence contains prefixes of two sequence.

(2) LCS has the Overlapping subproblem Property:-

To find LCS of X & Y, we may need to find LCS of X & Y_{n-1} and of X_{m-1} & Y in case of II. & III of the theorem. They both may need to find LCS of X_{m-1} & Y_{n-1}.

So, the LCS problem can be solved by D.P. For this we'll create a table C to store the lengths of LCS of various subproblems-solutions.

$c[i,j]$ = length of an LCS of the sequences X_i & Y_j

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

~~Ex: Algorithm for Implementation:-~~

* Algorithm LCS (X, Y, c)

```
{
  for i ← 0 to x_m do
  {
    for j ← 0 to y_n do
    {
      if { i = 0 or j = 0
           c[i,j] = 0;
         }
    }
  }

  for i ← 1 to x_m do
  {
    for j ← 1 to y_n do
    {
      if (x_i = y_i) and (x,y > 0)
      {c[i,j] = c[i-1, j-1] + 1
      }
      else
      {
        max c[i,j] = max(c[i,j-1], c[i-1],j)
      }
    }
  }
}
```

Ex(1) Find out the Longest common subsequence for the following:-

$X = \langle A, B, C, B, D, A, B \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

			B	D	C	A	B	A
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	3	3	4
B	7	0	1	2	2	3	4	4

$\therefore$ BCBA is the Longest common subsequence.

Soln $\Rightarrow c[1,1] \Rightarrow i=1, j=1$, $i,j>0$ and $x_i \neq y_i$

$\Rightarrow c[1,1] = c[i-1, j-1]+1 \Rightarrow c[1,1] = \max(c[i,j-1], c[i-1,j])$

$= c[1-1, 1-1]+1$ $\quad = \max(c[1,1-1], c[1-1,1])$

$= c[0,0]+1$ $\quad = \max(c[1,0], c[0,1])$

$= 1$ $\quad = \max(0,0)$

$= 0$

$c[1,2] = i=1, j=2$ and $i,j>0$ and $x_i \neq y_i$

$\Rightarrow c[1,2] = \max(c[1,2-1], c[1-1,2])$

$= \max(c[1,1], c[0,2])$

$= \max(0,0)$

$= \underline{0}$

Similarly, find all the values from $c[1,3]$ ..... to $c[1,6]$.

$\Rightarrow c[2,1] \Rightarrow i=2$, and $j=1$, $i,j>0$ and $x_i = y_j$

$c[2,1] \Rightarrow c[i-1, j-1]+1$

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

$= c[1,0]+1$

$= 0+1$

$= \underline{\underline{1}}$

$\Rightarrow c[2,2] \Rightarrow i=2$, and $j=2$, $i,j>0$ and $x_i \neq y_j$

$\therefore c[2,2] = \max(c[2,2-1], c[2-1,2])$

$= \max(c[2,1], c[1,2])$

$= \max(1, 0)$

$= \underline{\underline{1}}$

Similarly, find all the values of LCS in the table for $c[i,j]$.

$\Rightarrow$ <u>Algorithm for Back Tracking and implementation</u>

```
Function LCS(x[1....m], Y[1....n]) : array[0...m, 0...n],
{
  for i ← 0 to m do
  {
    for j ← 0 to n do
    {
      if i = 0 or j = 0
      {
        c[i,j] = 0
      }
    }
  }
```

Assembly Line Scheduling:

→ A factory has 2 assembly line $i = 1, 2$. Each assembly line has n stations. $j = 1, 2, 3 \ldots n$.

S_{ij} = j^{th} station on line n.

a_{ij} = assembly time required at station S_{ij}

e_i = entry time to enter assembly line i.

x_i = Exit time to exit assembly line.

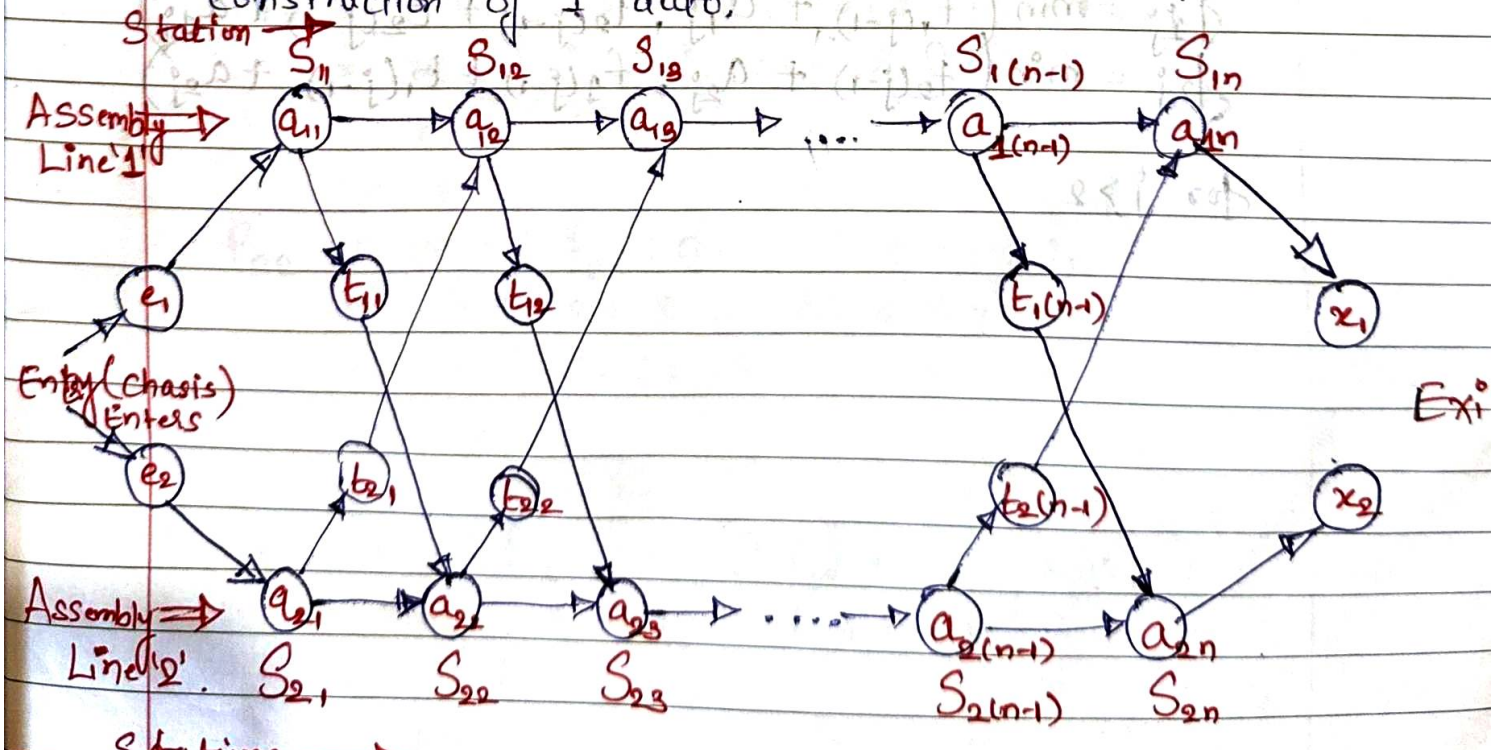
→ A chasis enters an assembly line. It passes through n stations. It can switch from 1 assembly line to the other after any station.

→ The time to go from one station to the next is

(I). Within the same assembly line $\cong 0$

(II) From assembly line i to the other line after station $S_{ij} = t_{ij}$, for $i = 1, 2$ and $j = 1, 2 \ldots n$ where t_{ij} is the transition time from station i to j.

Problem St. → The problem is to determine which station to choose from Line 1 and which to choose from Line 2 to minimize the total time through the factory for construction of 1 auto.



→ The brute force approach takes $O(2^n)$ time because choosing stations from line 1 is a subset of $[1, 2, \ldots n]$. There are 2^n subsets.

⟹ <u>Principle of Optimality holds</u> :-

Finding the fastest way through stations S_{ij} [Problem] contains an optimal solution to finding the fastest way through $S_{1,j-1}$ or $S_{2,j-1}$ (subproblems).

→ We can create f to store largest possible time t_{ij} = fastest possible time to get choosen from the starting point to station S_{ij}. We've to determine

$$f^* = \min(f_{1n} + x_1, f_{2n} + x_2)$$

We have:

$$f_{11} = e_1 + a_{11}$$

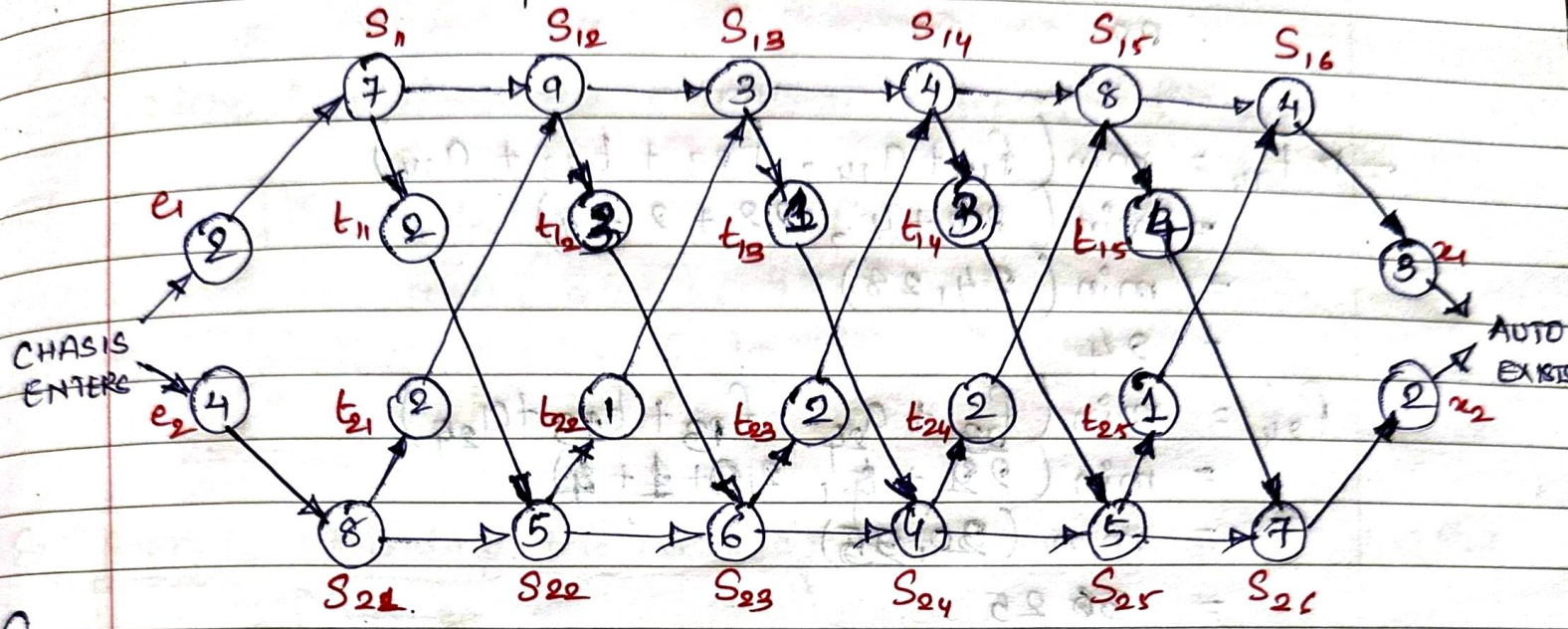
& $$f_{21} = e_2 + a_{22}$$

$$f_{1j} = \min(f_{1(j-1)} + a_{1j}, f_{2(j-1)} + t_{2(j-1)} + a_{1j})$$

$$f_{2j} = \min(f_{2(j-1)} + a_{2j}, f_{1(j-1)} + t_{1(j-1)} + a_{2j})$$

for $j \geq 2$

Ex① Find out the minimum time for completion of an auto using assembly line scheduling and also find out the shortest path.



Soln:-

$\Rightarrow f_{11} = e_1 + a_{11}$
$= 2 + 7 = 9$

$f_{21} = e_2 + a_{21}$
$= 4 + 8 = 12$

$\Rightarrow f_{12} = \min(f_{11} + a_{12},\ f_{21} + t_{21} + a_{12})$
$= \min(9 + 9,\ 12 + 2 + 9)$
$= \min(18, 23)$
$= 18$

$f_{22} = \min(f_{21} + a_{22},\ f_{11} + t_{11} + a_{22})$
$= \min(12 + 5,\ 9 + 2 + 5)$
$= \min(17, 16)$
$= 16$

$\Rightarrow f_{13} = \min(f_{12} + a_{23},\ f_{22} + t_{22} + a_{13})$
$= \min(18 + 3,\ 16 + 1 + 3)$
$= \min(21, 20)$
$= 20$

$f_{23} = \min(f_{22} + a_{23},\ f_{12} + t_{12} + a_{23})$
$= \min(16+6,\ 18+3+6)$
$= \min(22, 27)$
$= 22$

$\Rightarrow f_{14} = \min(f_{13} + a_{14},\ f_{23} + t_{23} + a_{14})$
$= \min(20+4,\ 22+2+4)$
$= \min(24, 28)$
$= 24$

$f_{24} = \min(f_{23} + a_{24},\ f_{13} + t_{13} + a_{24})$
$= \min(22+5,\ 20+1+5)$
$= \min(27, 26)$
$= 25$

$\Rightarrow f_{15} = \min(f_{14} + a_{15},\ f_{24} + t_{24} + a_{15})$
$= \min(24+8,\ 25+2+8)$
$= \min(32, 35)$
$= 32$

$f_{25} = \min(f_{24} + a_{25},\ f_{14} + t_{14} + a_{25})$
$= \min(25+5,\ 25+3+5)$
$= 30$

$\Rightarrow f_{16} = \min(f_{15} + a_{16},\ f_{25} + t_{25} + a_{16})$
$= \min(32+4,\ 30+1+4)$
$= \min(36, 35)$
$= 35$

$f_{26} = \min(f_{25} + a_{26},\ f_{15} + t_{15} + a_{26})$
$= \min(30+7,\ 32+4+7)$
$= \min(37, 43)$
$= 37$

⇒ Now creating a table of f_{1j} and f_{2j}

i \ j →	1	2	3	4	5	6
f_1	9	18	20	24	32	35
f_2	12	16	22	25	30	37

Now, computing the minimum time ie f^* as

$$f^* = \min(f_{1n} + x_1, f_{2n} + x_2)$$
$$= \min(35+3, 37+2)$$
$$= \min(38, 39)$$
$$\boxed{f^* = 38} \rightarrow \text{Minimum Time}$$

Sol^n PART-2 Finding the Path of Minimum Time (Fastest Path)

	j →	1	2	3	4	5	6
f_{1j} – l_1		–	1	2	1	1	2
f_{2j} – l_2		–	1	2	1	2	2

Now, to obtain the path, which gives fastest time in producing auto, we will use all the l_{ij} values

⇒ Algorithm

let $i = l^*$

print "Line i" station "n".

As $i=1$, we get first Message as Line 1 Station 6.

then, for ($j = n$; $j \geq 2$; j--)

{

$i = l_{i[j]}$

Print "Line i" station "$j-1$"

}

⇒ let $j = 6$, then $i = l_1[6] = 2$, Hence the message point will be Line 2 Station 5.

→ Let $j = 5$, then $i = l_1[5] = 2$, Hence Line 2 Station 4.

→ let $j = 4$, then $i = l_1[4] = 1$, Hence Line 1 Station 3

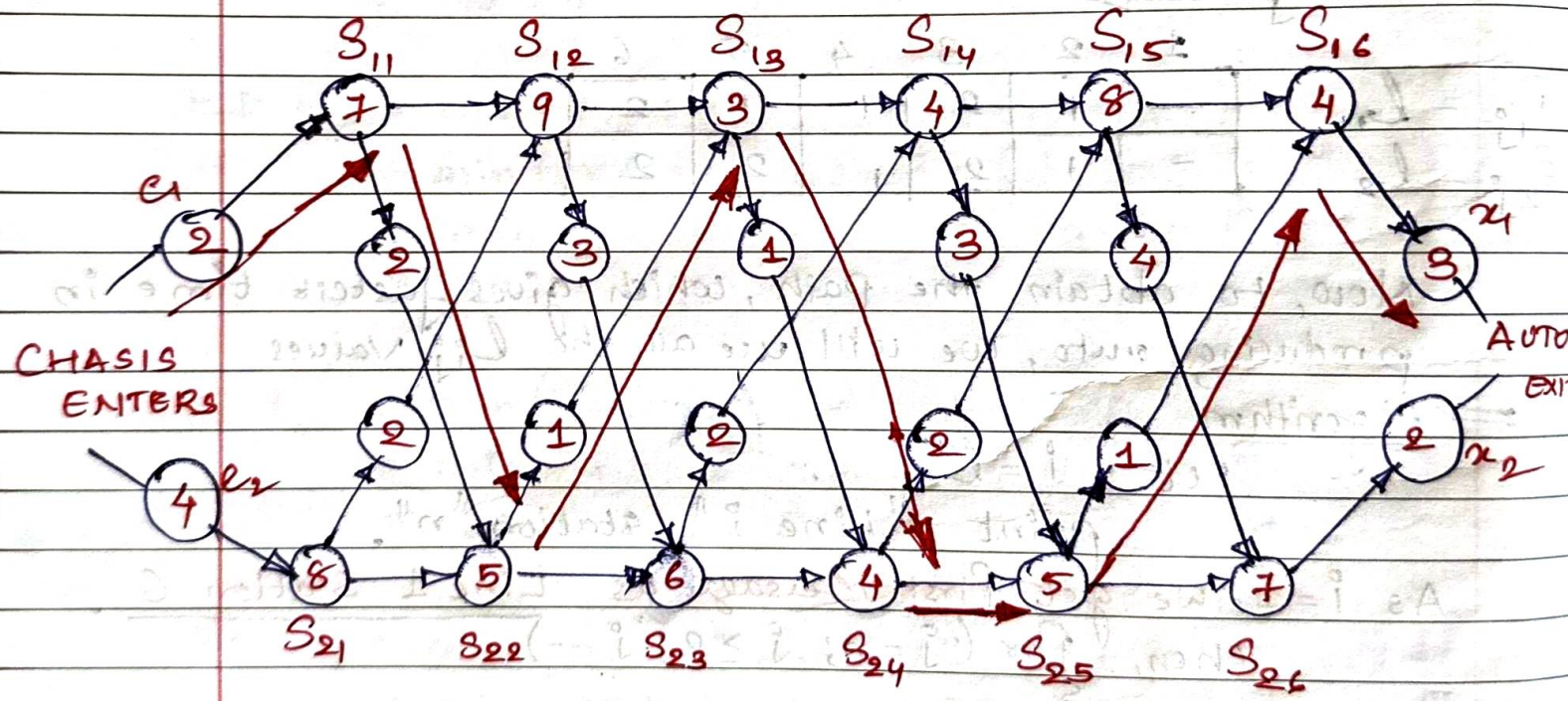
→ Let $j = 3$, then $i = l_1[3] = 2$. Hence Line 2 Station 2

→ Let $j = 2$, then $i = l_2[2] = 1$. Hence Line 1 Station 1.

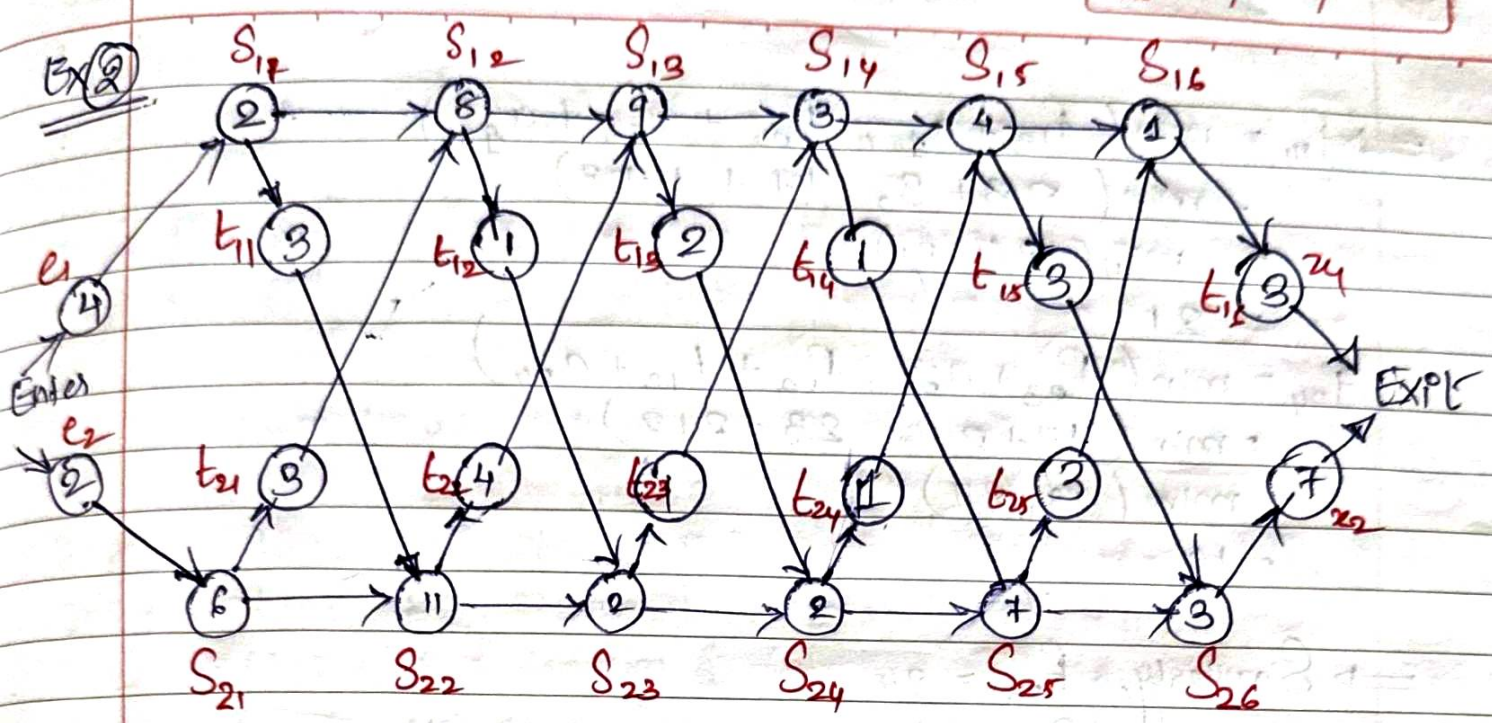
Hence, the Optimal Path in ALS will be

→ Line 1 Station 6
Line 2 Station 5
Line 2 Station 4
Line 1 Station 3
Line 2 Station 2
Line 1 Station 1

Hence Final Path is,



Ex (2)



Sol^n:-

$\Rightarrow f_{11} = e_1 + a_{11}$
$= 4 + 2 = 6$

$f_{21} = e_2 + a_{21}$
$= 2 + 6 = 8$

$\Rightarrow f_{12} = \min(f_{11} + a_{12}, f_{21} + t_{21} + a_{12})$
$= \min(6 + 8, 8 + 3 + 8)$
$= \min(14, 19)$
$= 14$

$f_{22} = \min(f_{21} + a_{22}, f_{11} + t_{11} + a_{22})$
$= \min(8 + 11, 8 + 3 + 11)$
$= \min(19, 22)$
$= 19$

$\Rightarrow f_{13} = \min(f_{12} + a_{13}, f_{21} + t_{22} + a_{13})$
$= \min(14 + 9, 19 + 4 + 9)$
$= \min(23, 32)$
$= 23$

$f_{23} = \min(f_{22} + a_{23}, f_{22} + t_{12} + a_{23})$
$= \min(19 + 2, 14 + 1 + 2)$
$= 17$

$\Rightarrow f_{14} = \min(f_{13} + a_{14}, f_{23} + t_{23} + a_{14})$
$= \min(23 + 3, 17 + 1 + 3)$
$= \min(26, 21)$
$= 21$

$f_{24} = \min(f_{23} + a_{24}, f_{13} + t_{13} + a_{24})$
$= \min(17 + 2, 23 + 2 + 2)$
$= \min(19, 27)$
$= 19$

$\Rightarrow$ Similarly, $f_{15} = 24$
$f_{25} = 26$

$\Rightarrow$ $f_{16} = 25$
$f_{26} = 29$

Now creating a table of f_{1j} and f_{2j} with minimum ~~line number~~ values.

	1	2	3	4	5	6
f_{1j}	6	14	23	21	24	25
f_{2j}	8	19	17	19	26	29

Now, calculating minimum time of computation is

$f^* = \min(f_{1n} + x_1, f_{2n} + x_2)$
$= \min(25 + 3, 29 + 7)$
$= \min(28, 36)$
$= 28$

Sol^n PART-II

Creating a table of f_{1j} and f_{2j} with minimum of line no.

$j \rightarrow$		1	2	3	4	5	6
f_1	l_1	–	1	1	2	2	1
f_2	l_2	–	2	1	2	2	2

Path is as follows:-

Line 1 Station 6

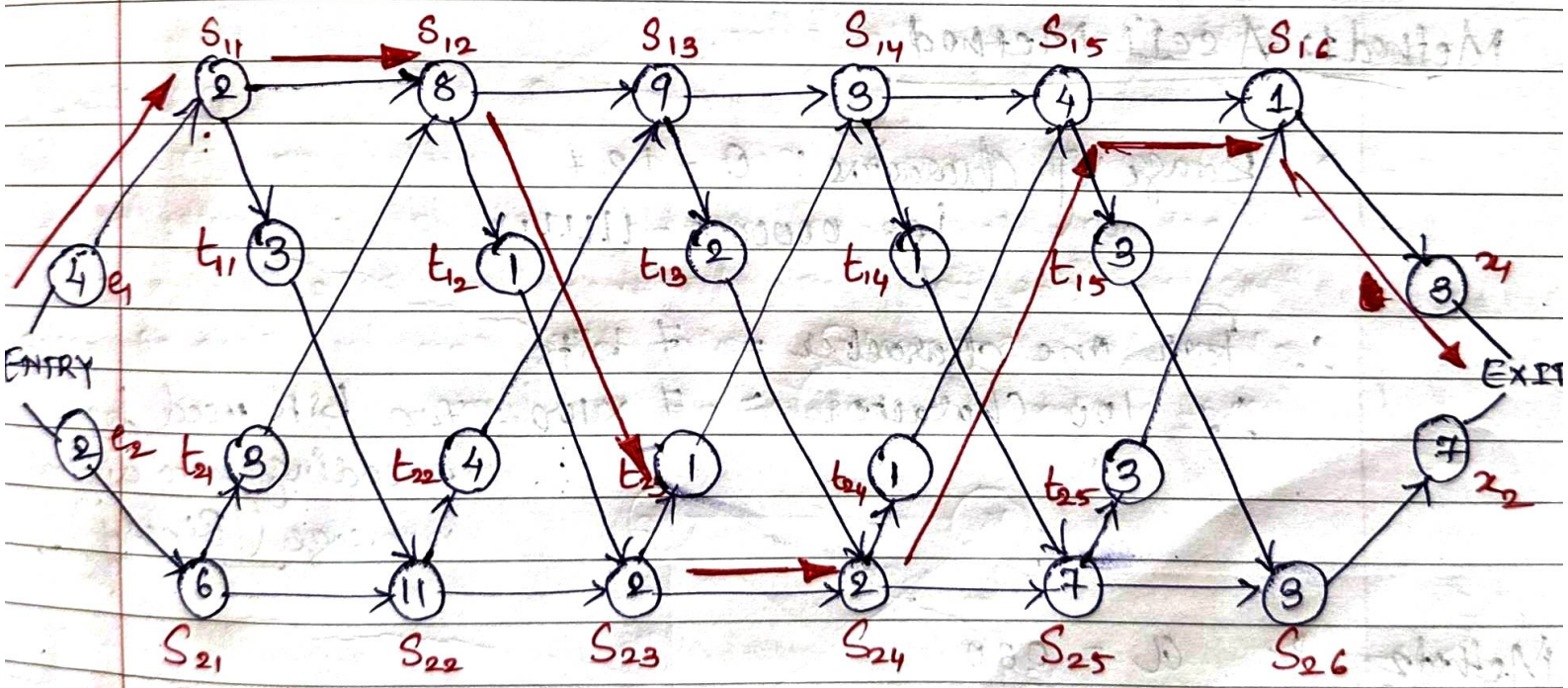
$j=6$, Line 1 Station 5 $i=1$

$j=5$, Line 2 Station 4 $i=2$

$j=4$, Line 2 Station 3 $i=2$

$j=3$, Line 1 Station 2 $i=1$

$j=2$, Line 1 Station 1



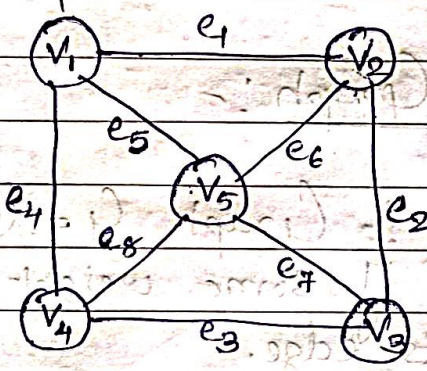
Unit - 4

DOMS Page No.
Date / /

Graph: Graph is non-linear, non-primitive data structure (Informal) used in many applications in computer science.
eg. Game playing, File system, shortest path, and many other problems are effectively formulated and solved using graph.

→ Graph G is defined by set of vertices (V) and set of Edges (E) joining those vertices. Graph is typically represented as ordered pair $G = (V, E)$.

eg



Graph $G = (V, E)$

→ Above graph G is described by the set of vertices $\{V_1, V_2, V_3, V_4, V_5\}$ and set of edges $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$

→ Vertices are referred as nodes of graph. Edges of graph are also referred as arcs or lines.

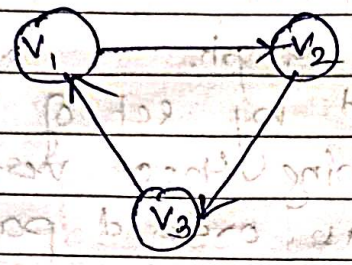
⇒ Categories of Graph

Divided into two Categories
1. Directed Graph
2. Undirected Graph

* **Undirected Graph:** Graph G is known as undirected graph if its edges do not have orientation.

→ Edges in Undirected Graph are bidirectional.

* **Directed Graph:** Graph G is said to be directed if its edges have some direction.
→ It is also known as digraph.



* **Properties of Graph:-**

(i) **Weighted graph:** Graph $G = (V, E, w)$ is called weighted graph if some weight or cost is associated with each of its edge.
→ In most of the cases, weights of edges are non-negative
→ Weighted graph can be directed or undirected.
→ Cost or weight of graph can be distance between two vertices or time taken from one vertex to other or amount of data that can be carried from one vertex to other at a time.

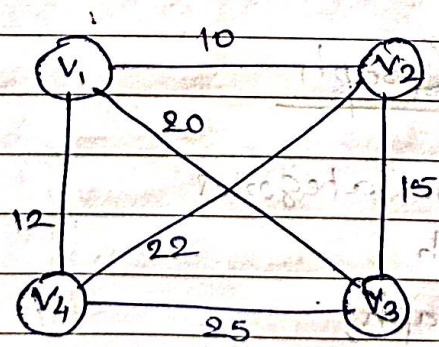
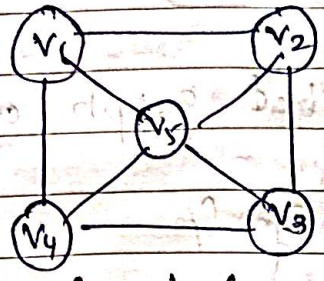


Fig. weighted Graph

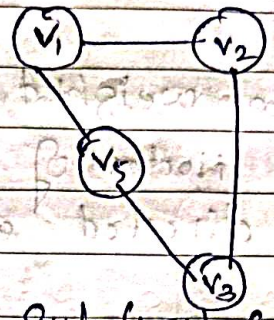
(ii) **Sub Graph:** Graph $G' = (V', E')$ is a sub-graph of graph $G = (V, E)$ if V' is the subset of V and E' is the subset of E.

→ Graph can have multiple sub-graphs

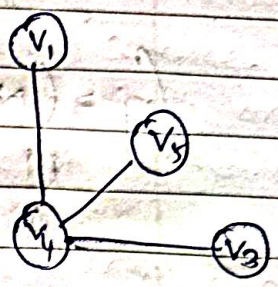
eg:



Graph G



Sub Graph. G_1



SubGraph G_2

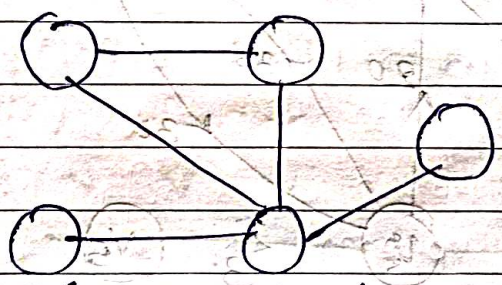
(iii) Sparse Graph :-

→ Graph $G = (V, E)$ is called sparse graph, if it contains very few edges.

→ Typically, in sparse graph, $|E| = O(|V|)$.

→ In sparse graph, number of edges are near to minimal number of edges. Sparse graph is efficiently represented using linked list.

→ eg:



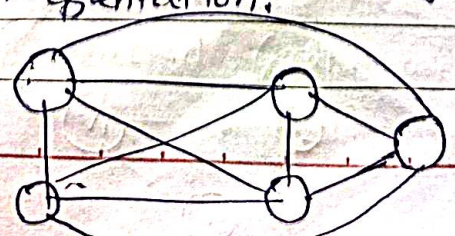
Sparse graph

(iv) Dense Graph :-

→ Graph $G = (V, E)$ is called dence graph, if no. of edges are near to maximum possible edges.

→ Typically, in dence graph, $|E| = O(|V^2|)$. Dense graph is best described by ~~matrix~~ adjacency matrix representation.

eg

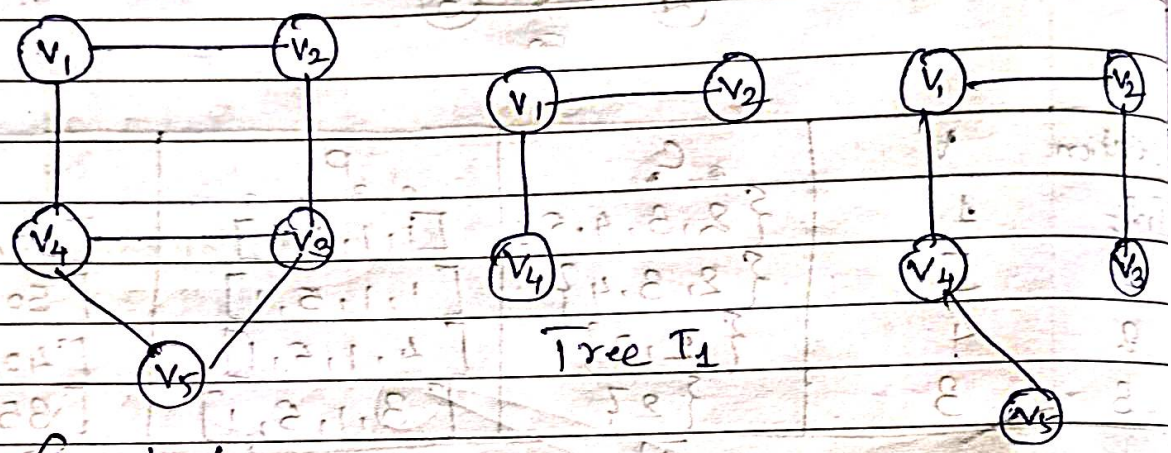


(v) **Tree** :- Tree $T = (V', E')$ is a subset of graph $G = (V, E)$, where V' is a subset of V and E' is a subset of E.

→ Tree does not contain any cycle, while graph or subgraph may contain cycle.

→ Graph can have multiple trees.

eg:



Graph G

Tree T_1

(vi) **Spanning Tree** :

→ Spanning Tree $T = (V', E')$ is a tree of connected, undirected, weighted graph $G = (V, E, w)$, which contains all the vertices of G and some or all edges of graph G. So $V' = V$ and $E' <= E$.

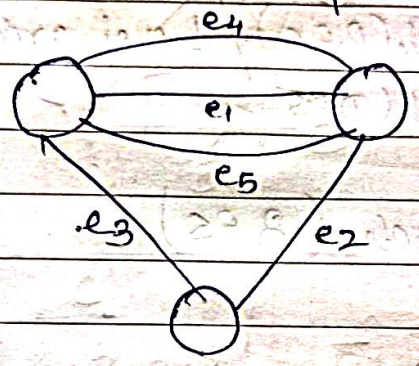
→ It does not contain cycle

(viii) **Minimum Spanning Tree** - is a spanning tree with minimum weight.

→ Kruskal's and Prim's have proposed efficient algo. to find minimum spanning tree of the graph.

(viii) Multi Graph:-

→ A graph having multiple edges between same pair of vertices is called multigraph. Such edges are also known as parallel edges.

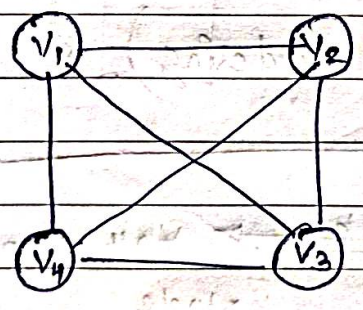


Multi Graph

(ix) Complete Graph:- Undirected graph G is called complete graph if each of the vertex of graph is connected with remaining all other vertices.

→ Complete graph with n vertice has $n*(n-1)/2$ edges.

eg:-

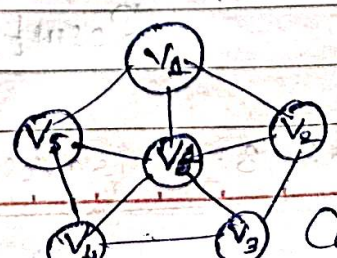


Complete graph

(x) Connected Graph: Undirected graph G is called connected if there exists a path between every pair of vertices (V_i, V_j)

→ In connected graph, we can reach to any vertex V from any other vertex u.

eg:



Connected Graph.

Minimum Spanning Tree Algorithms

(1) Prim's Algorithm:-

Problem Statement: Find the minimum spanning tree with minimum weight.

Steps:

i) choose an arbitrary node

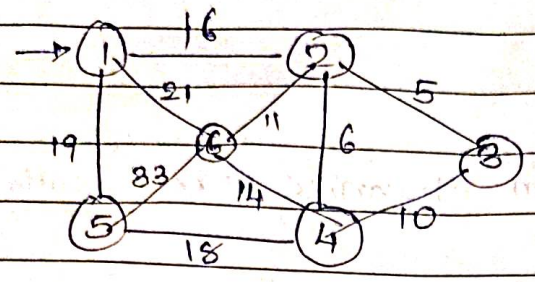
ii) at every stage, select the shortest edge that can extend the tree to an additional node.

iii) repeat it till we have n vertices in T.

Algo:

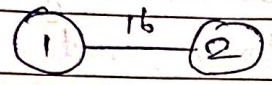
```
Function_MCT_PRIM( G=<N,A>: graph , length: A → R+: set of edges)
{
    T ← ϕ
    B ← { an arbitrary Node of N }
    While B ≠ N do
    {
        Select (u,v) ← shortest edge such that
                        (u∈B) and v∈(N−B)
        T ← T ∪ {(u,v)}
        B ← B ∪ {v}
    }
    return T
}
```

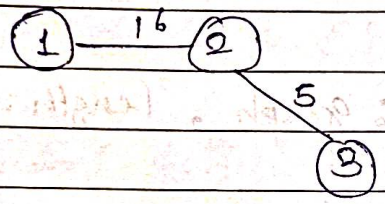
eg (1)

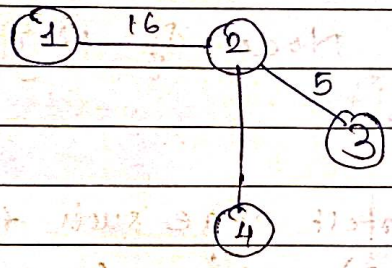


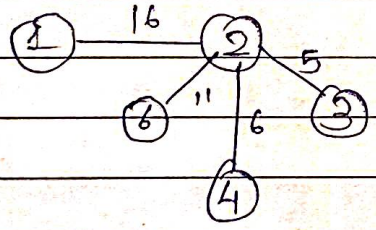
Start Node = 1

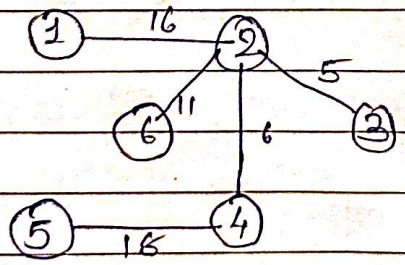
⟹











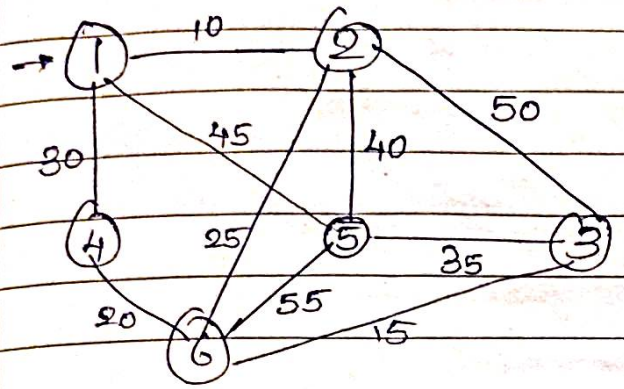
⟹ ∴ Solution MST = $\{(1,2),(2,3),(2,4),(2,6),(4,5)\}$

MST value = <u>56</u>

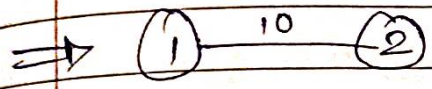
Date / /

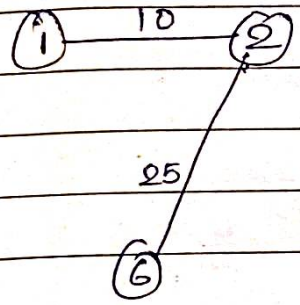
Page No.

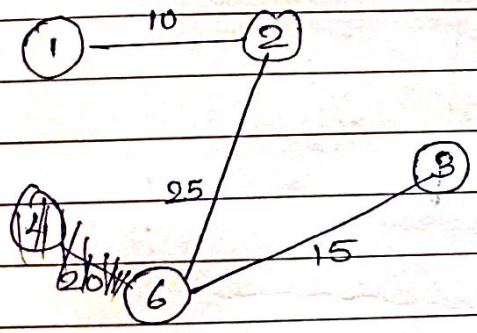
eg (2)

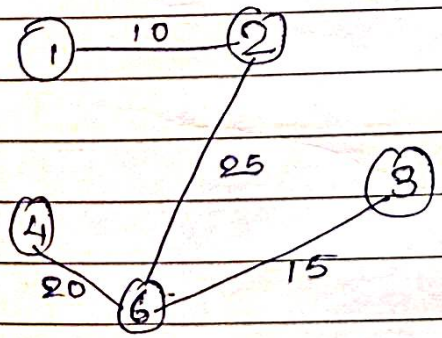


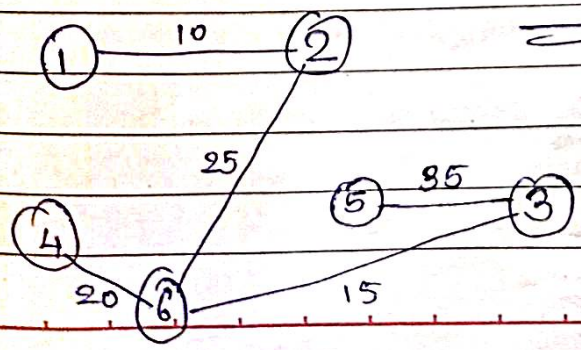
Start Node = 1.











$\Rightarrow$ Solution MST = $\{(1,2), (2,6), (6,3), (6,4), (3,5)\}$

MST value = $\underline{\underline{105}}$.

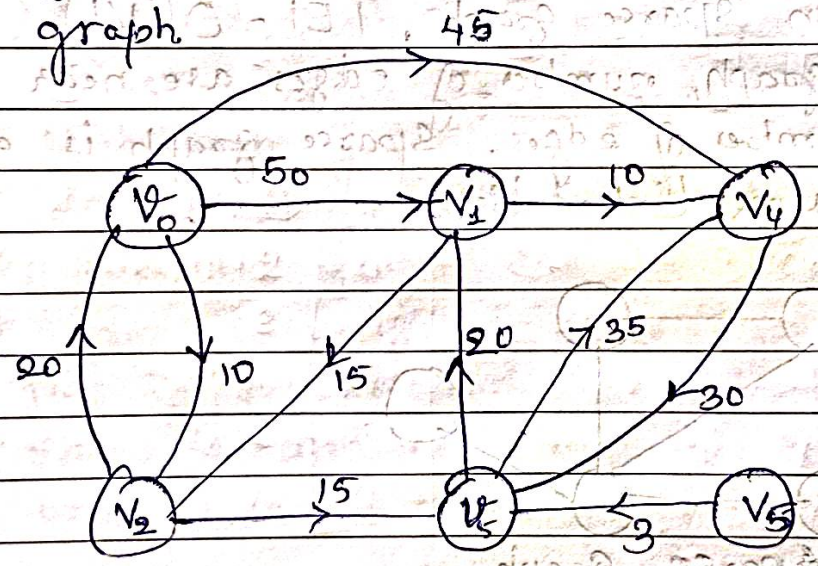
* Single Source Shortest Path Algorithm :-

$G = (N, A)$ is a weighted directed graph where
N = set of Nodes of G
A = set of directed edges of G

→ Each edge has a non-negative length.
→ One of the node is considered as source node.

Problem statement – The Problem is to determine the shortest Path from the source to all other nodes of the graph.

eg



Source Node = V_0

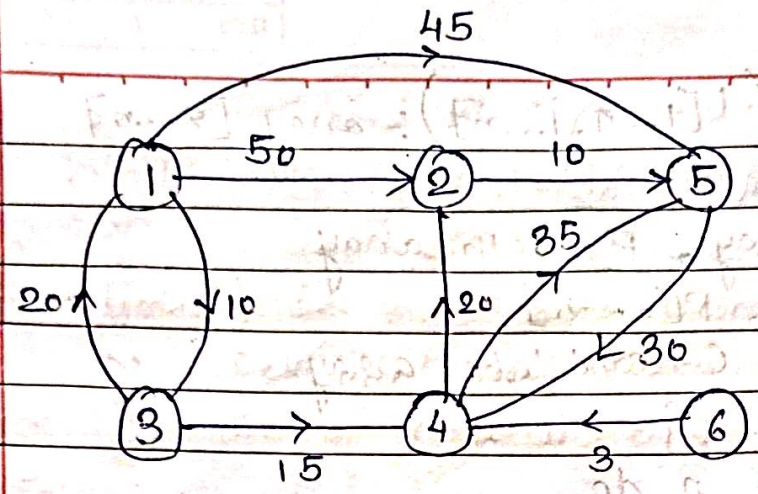
DOMS Page No.
Date / /

* Algorithm:-

```
function Dijkstra(L[1....n, 1....n]): array [2....n]
{
▷ D - distance array, P - path array
  array D[2.....n], P[2....n]
▷ initialization of C - candidate array.
  C ← {2, 3, .... n}
  for i ← 2 to n do
  {
      P[i] ← 1
      D[i] ← L[1, i]
  }
  repeat (n-2) times
  {
  V ← an element of C having minimum D[v]
  C ← C - {v}

  for each w ∈ C do
      if D[w] > D[v] + L[v, w] then
      {
      P[w] ← v
      D[w] ← D[v] + L[v, w]
      }
  return P
}
```

Ex(1)



Iteration	V	C	P (2, 3, 4, 5, 6)	D (2, 3, 4, 5, 6)
Initial	1	{2, 3, 4, 5, 6}	[1, 1, 1, 1, 1]	[50, 10, ∞, 45, ∞]
1	3	{2, 4, 5, 6}	[1, 1, 3, 1, 1]	[50, 10, 25, 45, ∞]
2.	4	{2, 5, 6}	[4, 1, 3, 1, 1]	[45, 10, 25, 45, ∞]
3.	2	{5, 6}	[4, 1, 3, 1, 1]	[45, 10, 25, 45, ∞]
4.	5	{6}	[4, 1, 3, 1, 1]	[45, 10, 25, 45, ∞]

1

Path Array	
2	4
3	1
4	3
5	1
6	1

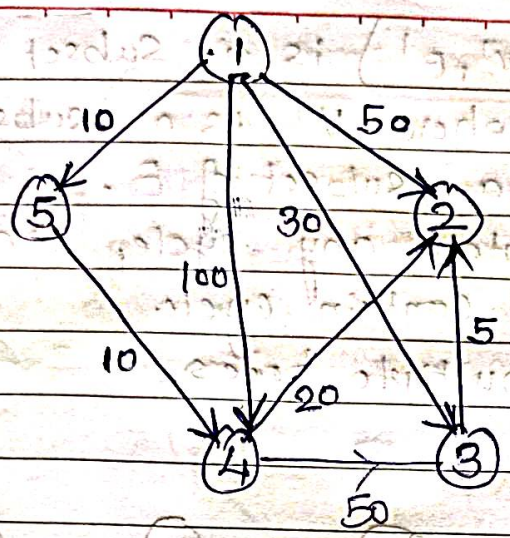
1

Distance	
2	45
3	10
4	25
5	45
6	∞

⟹ So, in order to reach Node 4 from 1, Path 4 ← 3 - 1 and distance d = 25

⟹ Similarly, you can ~~reach~~ find shortest distance to reach any node from Node 1. and and its path.

Ex (2)



Iteration	V	C	P (2, 3, 4, 5)	D (2, 3, 4, 5)
Init	1	{2, 3, 4, 5}	[1, 1, 1, 1]	[50, 30, 100, 10]
1	5	{2, 3, 4}	[1, 1, 5, 1]	[50, 30, 20, 10]
2	4	{2, 3}	[4, 1, 5, 1]	[40, 30, 20, 10]
3	3	{2}	[3, 1, 5, 1]	[35, 30, 20, 10]

1

Path array:

2	3
3	1
4	5
5	1

1

Distance Array:

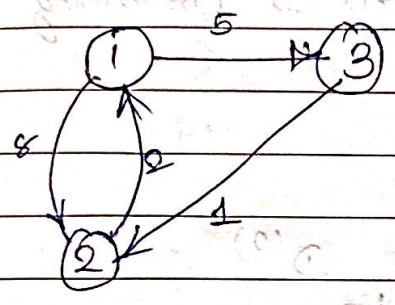
2	35
3	30
4	20
5	10

Floyd's Warshall Algorithm

→ Also known as all pair shortest path algorithm.

→

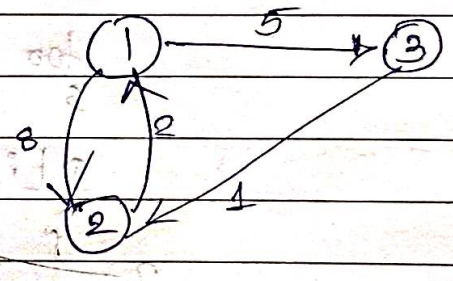
eg ①



Soln ⇒ First we Compute

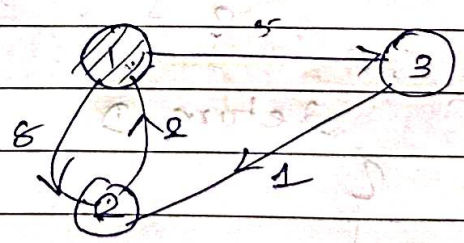
$D^{(0)}$

	1	2	3
1	0	8	5
2	2	0	∞
3	∞	1	0



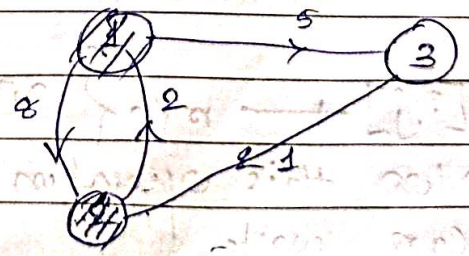
$D^{(1)}$

	1	2	3
1	0	8	5
2	2	0	7
3	∞	1	0



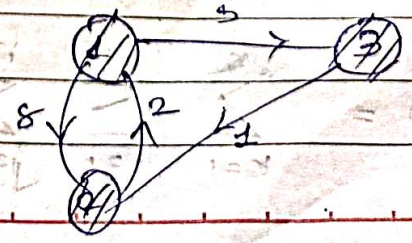
$D^{(2)}$

	1	2	3
1	0	8	5
2	2	0	7
3	3	1	0



$D^{(3)}$

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



Algorithm :-

```
Algo Floyd_Shortest_path (wt[1....n, 1....n])
{
    // Input: weighted Matrix wt[1....n], [1....n] for given
            graph
    // Output: The distance Matrix D containing shortest
            paths.

⇒ D ← wt
    // This initialization gives D^(0)
    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
}
```

Analysis

$D[i,j] \leftarrow \min\{D[i,j], D[i,k]+D[k,j]\}$

Now this operation is in three nested loops, we can write

$$c(n) = \sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} 1$$

$$\Rightarrow C(n) = \sum_{k=1}^{n} \sum_{i=1}^{n} (n-1+1) \qquad \because \sum_{i=l}^{u} 1 = u-l+1$$

$$= \sum_{k=1}^{n} \sum_{i=1}^{n} n$$

$$= \sum_{k=1}^{n} n^2 \qquad \boxed{\because \sum_{i=1}^{n} i = \frac{1}{2} n^2, \text{ neglect constant}}$$

$$= n^3 \qquad \boxed{\because \sum_{i=1}^{n} n^2 = \frac{1}{3} n^3, \text{ neglect constant}}$$

$\therefore$ The time complexity of finding all pair shortest path is $\underline{\underline{O(n^3)}}$.

* Solving the same example using formula

$D^0 =$

	1	2	3
1	0	8	5
2	2	0	∞
3	∞	1	0

Formula :-

$$D^k[i,j] = \min\{D^{k-1}[i,j], D^{k-1}[i,k] + D^{k-1}[k,j]\}$$

$\Rightarrow$ for $D^{(1)} = D$

$D^1 =$

	1	2	3
1	0	8	5
2	2	0	7
3	∞	1	0

$$D^1[2,3] = \min\{D^0[2,3], D^0[2,1] + D^0[1,3]\}$$
$$= \min\{\infty, 7\}$$
$$= 7$$

$$D^2[3,1] = \min\{D^1[3,1],\ D^1[3,2] + D^1[2,1]\}$$
$$= \min\{\infty,\ 3\}$$
$$= \underline{\underline{3}}$$

Similarly, you can calculate rest of the values and obtain final answer

$D^3 =$

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

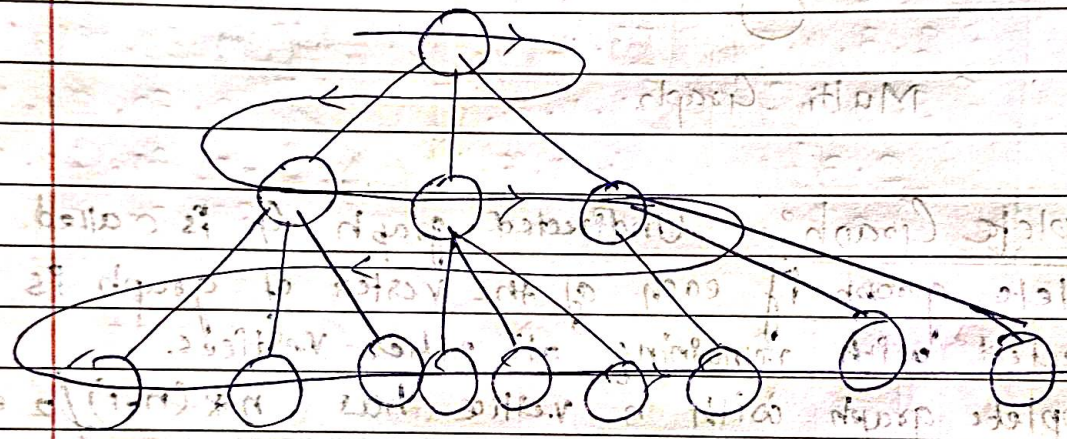
Dept

* Introduction to Graph Traversal :

→ The process of visiting and exploring a graph for processing is graph traversal.

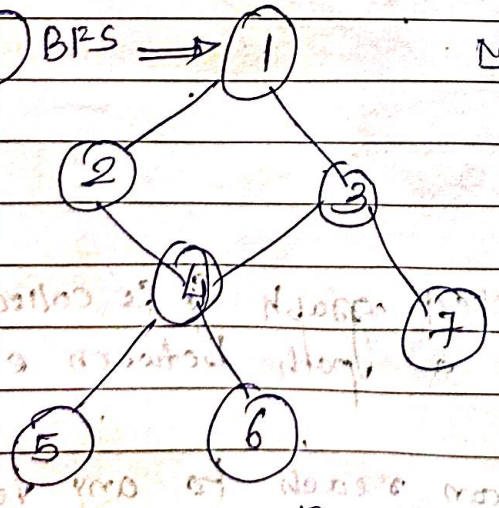
1. Breath First Search (BFS) / Level order Traversal
2. Depth First Search (DFS).

1 Breath First Search (BFS)



⇒ level by level (level ordering)

eg (1) BFS ⇒



Note → You can start from any Node

→ Data structure to be used is queue.

→ BFS can be multiple.

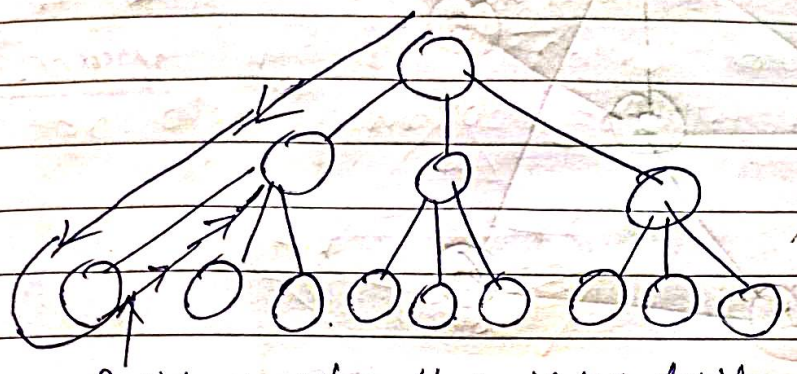
1	2	3	4	7	5	6

Result :- 1 2 3 4 7 5 6

Final BFS Traversal ⇒

Time Complexity = O(V+E)

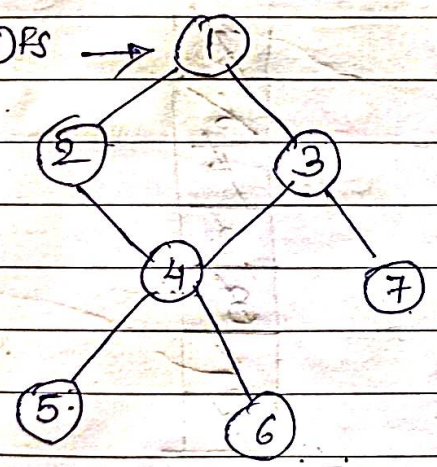
(2) Depth First Search (DFS)

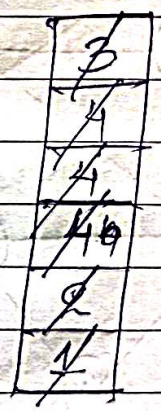


Backtrack when there is no depth

⇒ Here it requires back tracking.
→ Data structure that is used here is stack.
→ Back track only where there is no option for going in depth.

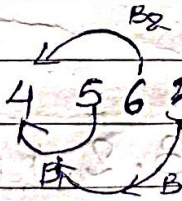
eg (1) DFS →





Complexity $= O(V+E)$ as we are travelling every vertex and edges..

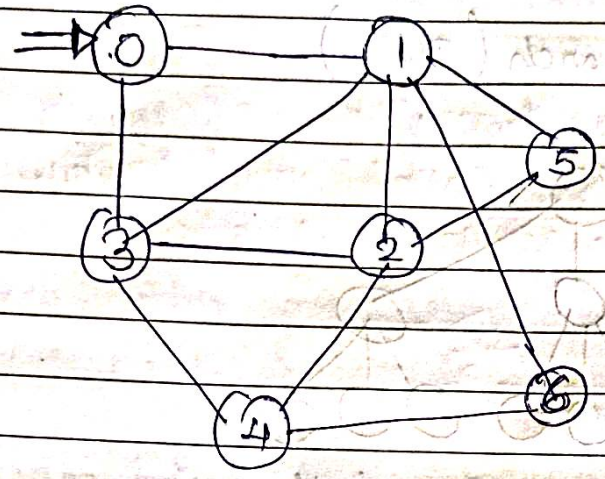
Result:- 1 2 4 5 6 3 7



⇒ Application

Web Crawlers, Social Media, Networking, Check whether there is cycle in graph or not, MST. etc → we can use BFS and DFS.

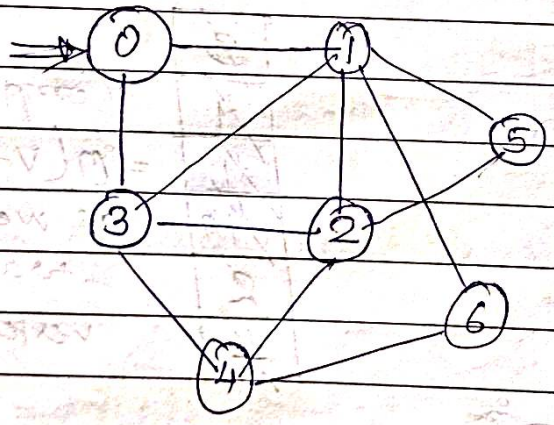
eg (2) BFS



(FIFO): Queue: | ~~0~~ | ~~1~~ | ~~3~~ | ~~2~~ | ~~5~~ | ~~6~~ | ~~4~~ |

Result:- 0 1 3 2 5 6 4

eg (2) DFS



~~5~~
~~6~~
~~4~~
~~2~~
~~3~~
~~1~~
~~0~~

Result:- 0, 1, 3, 2, 4, 6, 5

BFS and DFS Theory

DOMS Page No.
Date / /

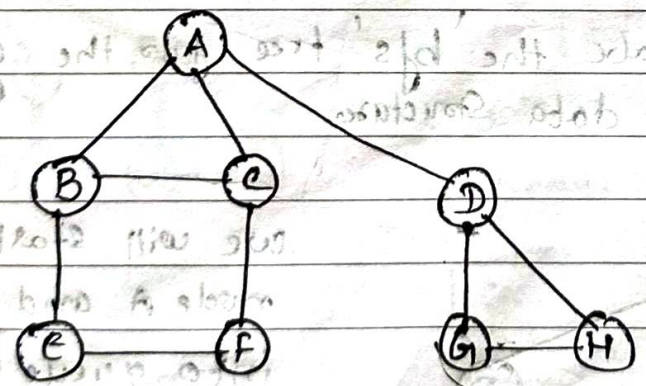
* Explain Breath First Search algorithm with the help of Example.

→ Searching a graph means traversing a graph from given vertex.

→ The BFS follows the following rule.

1. Select an unvisited node V, visit it, and make the root of the tree formed. Its level is called the current level
2. From each node X in the current level, the order in which all the level nodes ~~are~~ were visited, visit all the unvisited neighbours of X.
   → The newly visited nodes from this level forms a new level. This new level becomes the next current level
3. Repeat the step-2 for all the unvisited nodes/vertices.
4. Repeat from step-1 until no more vertices are remaining.

Now Consider the graph

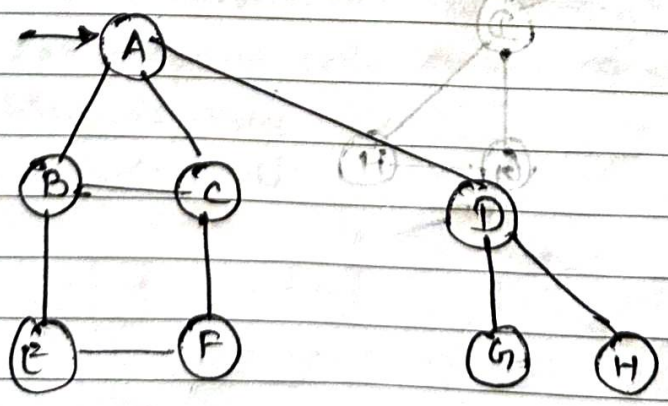


```
Algorithm BFS(G)
{
  // Problem Description: This algo. is for finding BFS.
  Queue Q.
  while ( G has unvisited node) do
  {
      V ← an unvisited node;
      visit[V] ← 1;
      eq_que(V,Q)
      while (Q is not empty) do
      {
          X ← def_que(Q)
          for( unvisited neighbour Y of X) do
          {
              visit[Y] ← 1
              en_que(Y,Q)
          }
      }
  }
}
```

Now we will generate the 'bfs' tree for the given graph

→ BFS uses Queue data structure

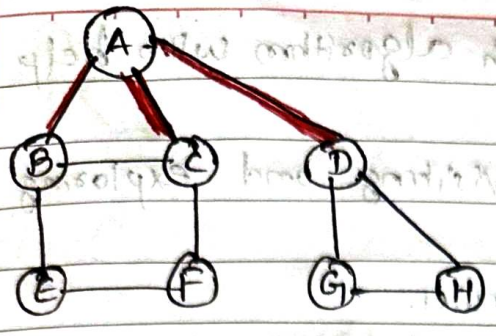
Step 1:



We will start search from node A and insert it into queue Q.

[A]

Step 2:



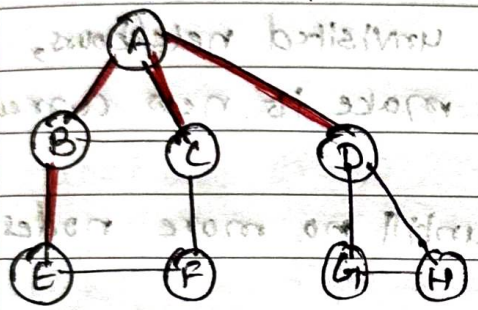
- Adjacent nodes to A will be obtained and inserted in the queue.
- Delete A from queue and print it

B	C	D

Node - A - deleted.

Result :- A

Step 3

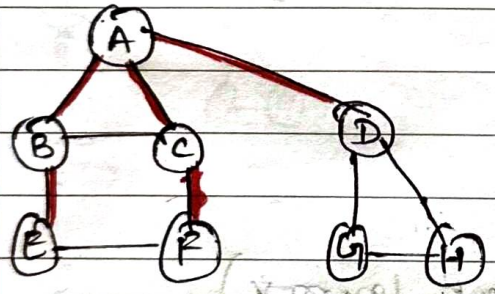


C	D	E

Node - A, B deleted

Result - A, B

Step 4:

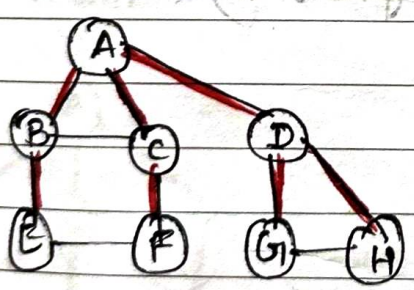


D	E	F

Node: A, B, C. deleted

Result : - A, B, C

Step 5:-

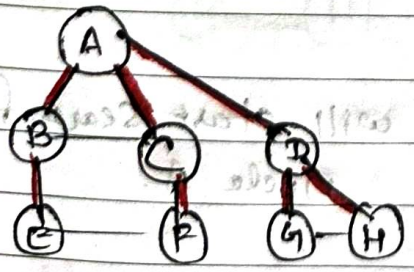


E	F	G	H

Node: A, B, C, D deleted

Result : - A, B, C, D

Step 6



→ Empty Queue

∵ there is not unvisited nodes remaining we will delete all the vertices from queue and print

Node: - A, B, C, D, E, F, G, H deleted

Result: - A, B, C, D, E, F, G, H

* Explain the Depth-First Search algorithm with help of example.

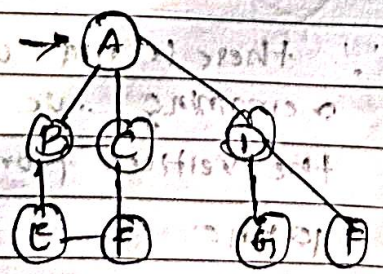
→ Searching a graph means visiting and exploring a vertex.

→ DFS follow the following rules

1. Select an unvisited node v, visit it and treat it as the current root node.
2. Find the unvisited neighbour of the current node, visit it and make it the new current node.
3. If the current node has no unvisited neighbours, backtrack to its parent and make it new current node.
4. Repeat the steps 2 and 3 untill no more nodes are left to be visited.
5. Repeat from step 1 for the remaining nodes.

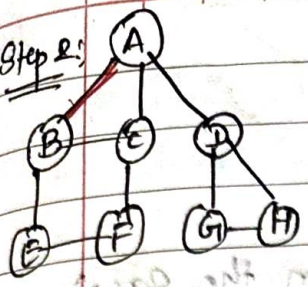
```
Algorithm DFS(v)
{
    visit[v] = 1
    for (each vertex X adjacent from v)
    {
        if (visit[X] = 0) then
            DFS(X)
    }
}
```

Example step 1 :

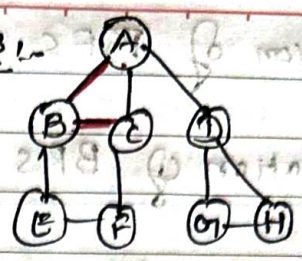


we will start search from vertex A.

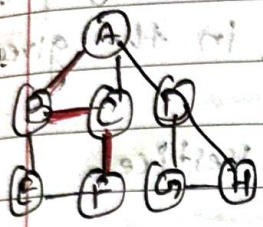
Step 2:



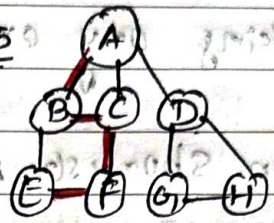
Step 3:



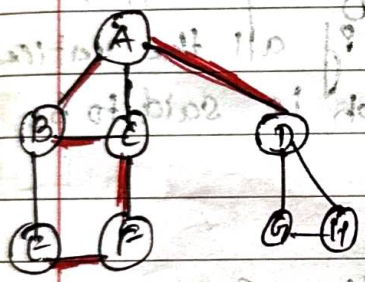
Step 4:

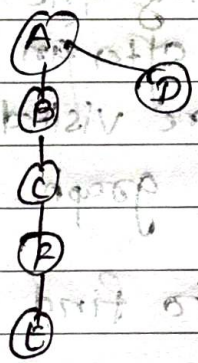


Step 5

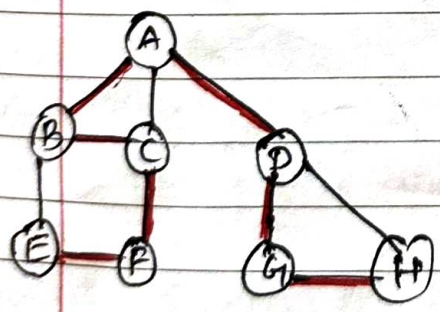


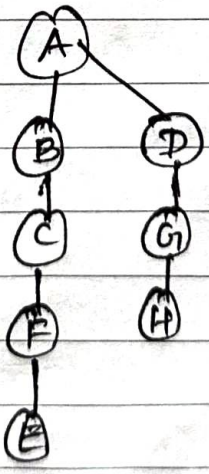
Step 6 - Now, there is no reachable nodes from E, so, we will backtrack and proceed.





Step 7: Same way we will visit G and then H.





Hence we obtain DFS Sequence as A, B, C, F, E, D, G, H.

* Application of BFS and DFS

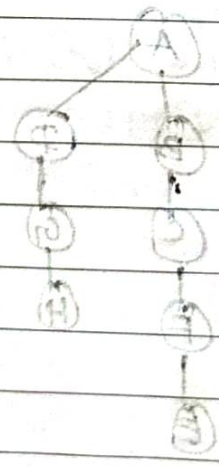
⟹ Application of BFS

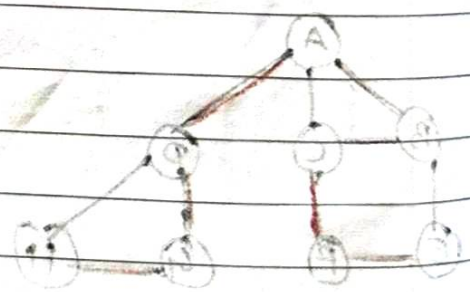
1. For finding the connected components in the graph.
2. For checking any existence of cycle in the given graph
3. To obtain shortest path between two vertices.

⟹ Application of DFS

1. DFS is used for checking connectivity of a graph.
   → Start traversing the graph using depth first method and after an algo. halts, if all the vertices of graph are visited then graph is said to be a connected graph.

2. DFS is used to find an articulation point.





String Matching

- String matching operation is core part in many text processing applications.
- Objective of string matching algorithm is to find pattern P from given Text T.

- String Matching problem is defined as follow:
  "Given some text string $T[1,\ldots,n]$ of size n, find all occurences of pattern $P[1\ldots m]$ of size m in T."
- We say that P occurs in text T with number of shifts s, if $0 \le s \le n-m$ and $T[(s+1)\ldots(s+m)] = P[1\ldots m]$

eg.

	1	2	3	4	5	6	7	8	9	10	11	12
Text T →	H	O	W		A	R	E		Y	O	U	?

Pattern P $\xrightarrow{s=4}$ | A | R | E |

Pattern matching in Text T

- In above example, pattern P = ARE is found in Text T after four shifts.
- Classical application of string matching algorithm is to find particular protien pattern in DNA sequence.

* Naive String Matching Algorithm:-

- This is simple and inefficient brute force approach.
- It compares first character of pattern with searchable text. If match is found, pointers in both strings are advanced. If match is not found, pointer of Text is incremented and pointer of pattern is reset. This process is repeated till end of the text.
- Naive approach does not require any preprocessing.

→ Given Text T and Pattern P, it directly starts comparing both strings character by character.

→ After each comparison, it shifts pattern string one position to the right.

Eg Step1

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T	P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

Here $T[1] \neq P[1]$

	1	2	3
P	A	N	D

Step2 $T[2] \neq P[1]$, so advances text pointer i.e. t_{i++}

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T	P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

	1	2	3
P	A	N	D

Step3 $T[3] = P[1]$, so it advances both pointers i.e. t_{i++}, P_{j++}

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T	P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

	1	2	3
P	A	N	D

Step4 $T[4] = P[2]$, so both advances its pointers i.e. t_{i++}, P_{j++}

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T	P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

	1	2	3
P	A	N	D

Step 5 $T[5] \neq P[3]$, so advance text pointer i and reset pattern pointer; ie. $t_i = (m-1)$, $P_j = 1$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

1	2	3
A	N	D

Step 6

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

1	2	3
A	N	D

$T[4] \neq P[1]$, so advance text pointer t_{i++}

Step 7 $T[5] \neq P[1]$, so advance text pointer (t_{i++})

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

1	2	3
A	N	D

Step 8 $T[6] \neq P[1]$, so advance text pointer ie. t_{i++}

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

1	2	3
A	N	D

Step 9 $T[7] \neq P[1]$, so advance text pointer ie. t_{i++}

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

1	2	3
A	N	D

Step 10 ↓

P.T.O

Step 10 T[8] = P[i], so advance text and Pattern Pointer i.e. t;++, P;++

P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

A	N	D

Step 11 T[9] = P[2], so advance text and Pattern Pointer i.e. t;++, P;++

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

A	N	D
1	2	3

Step 12 T[10] = P[3], so advance text and Pattern, if Pattern Not Match. i.e. t;++, P;++.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I	S

A	N	D

Pattern Match / Found

Eg. 2 RAMAN LIKES MANGO — Text

MANGO — Pattern

* Algorithm:-

```
Algorithm NaiveSM(T[0....n-1], P[0....m-1])
{
  for i ← 0 to n-m do
  {
    j ← 0
    while j < m and P[j] = T[i+j] do
      j ← j+1
    if (j == m)
      return i;
  }
  return -1
}
```

* Complexity Analysis

Best → In best case, searchable text does not contain any of the prefix of pattern. Only one comparison requires moving pattern one position right.

Example:- T = ABABCDBCAC , P = XYXZ.

Algorithm does $O(n-m)$ comparison ⟹ $O(n)$

Worst & Average → In worst case, first $(m-1)$ characters of pattern and text are matched and only last character does not match.

Example:- T = AAAAAAAAAAC , P = AAAAB.

Algorithm takes $O(m*(n-m))$ ~~comp~~ time ⟹ $O(m \cdot n)$

→ Also, average case time complexity is same as that of worst case ⟹ $O(m \cdot n)$

* Rabin-Karp Algorithm – by Michael O. Rabin & Richard M. Karp.

→ The Rabin Karp method is based on hashing technique and brute force for comparison.

→ This algorithm assumes each character to be a digit in radix - d notation.

→ It makes use of equivalence of two numbers modulo.

→ Let $P[1 \ldots m]$ be a pattern, then P denotes its corresponding decimal value.

→ Takes $d = 10$ for decimal number and $d = 2$ for binary number

→ Similarly, let $T[1 \ldots n]$ to be a Text, then t_s denotes the decimal value of substring $T[s+1 \ldots s+m]$ for $s = 0, 1, \ldots n-m$.

→ For computing t_{s+1} is given by,

$$t_{s+1} = d\left(t_s - d^{m-1} T[s+1]\right) + T[s+m+1]$$

where $d = 10$ (decimal).

Example 1: Let $T = 3\,1\,4\,1\,5\,9\,2\,6\,5\,3\,5\,8\,9\,7\,9\,3$ and Pattern $P = 26$. with modulo $q = 11$.

⇒ Let,

Text	3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
Pattern	2	6														

Step 1: Pattern modulo q

⇒ $26 \bmod 11 = \underline{4}$

Step 2. We will now obtain mod 11 for each text T with substring of size m.

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

9	3	8	4	4	4	4	10	9	2	3	1	9	2	5

(Spurious hits) (ie. not actual pattern match)

→ Valid Match (hit)

→ For $m=2$, ($\because$ Pattern $P=26$ to match has 2 digits or characters) $t_s = 26$, then we can compute t_{s+1} as follows:

[Old Highest order bit] [New low order bit]

$$t_{s+1} = 10\left(t_s - 10^{m-1} \cdot T[s+1]\right) + T[s+m+1]$$

$$= 10\left(26 - 10 \cdot 2\right) + 5$$

$$= 10(6) + 5$$

$$= 65$$

Thus we can obtain every successive bit in Text T. Now lets say we got spurious hits and hit but to find out which one is spurious and which one is actual, can be obtained by successive check of bits.

Text T

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

Hash values

8	4	4	4	4

↑ To check whether this is spurious or actual.

P.T.O

Now, Find the actual value of all hits to identify spurious and actual hits.

Let $t_s = 41$

$\therefore t_{s+1} = 10(41 - 10 \cdot T[s+1]) + T[s+m+1]$

$= 10(41 - (10 \cdot 4)) + 5$

$= 10(1) + 5$

$= 15 \neq 26$ $\therefore$ It is a spurious hit as it does not matches the pattern.

Next

$t_{s+2} = 10(t_{s+1} - 10\, T[s+1]) + T[s+m+1]$

$= 10(15 - 10 \cdot 1) + 9$

$= (10 \times 5) + 9$

$= 59 \neq 26$ $\therefore$ It is a spurious hit as it does not matches the pattern.

→ Similarly, ~~it matches~~ you can find all hits and get your answer of actual hit.

* Complexity Analysis:-

→ Rabin karp algorithm is randomized algorithm. In most of the cases, it runs in linear time, i.e. $O(n)$.

→ However, worst case of rabin karp algorithm is as bad as naive algorithm i.e. $O(mn)$ but its very rare.

→ It can happen only when prime number used for hashing is very small.

* Knuth-Morris Pratt algorithm

→ This is the first linear time algorithm for string matching.
→ This approach keeps track of matched part of pattern
→ Main idea of algorithm is to avoid computation of transition function δ and reducing useless shifts performed in naive approach.
→ By maintaining this information of already processed string, this approach reduces the time O(m+n). That is, in worst case, KMP algorithm examines all character of input pattern and text exactly once.
→ This improvement is achieved by using auxillary function π.
→ Computation of auxillary function π take O(m) time.

* Steps to compute Prefix function.

Step 1:- Create a prefix function ~~table~~ with index 1, as follows. eg. a b a b d ← Pattern. with a table.

i	0	1	2	3	4	5
P(i)		a	b	a	b	d
π(i)		0	0	1	2	0

Step 2 Fill up all the values inside the table by counting the prefix and following the algo. of prefix.

Note:→ Prefix π table generation is explained later on over here, NO algorithm of prefix function required.

Comparison with Naive String Matching

Consider

Text :- a b c d a b c a b c d f

1 2 3 4 5 6 7 8 9 10 11 12

j

1 2 3 4 5

Pattern :- a b c d f

→ Compare Text[i] and Pattern[j], if matching move/shift i and j by 1 towards Right.

→ If no match, reset j and move i to next starting substring.

Problem: Now i is shifting/doing Backtracking which is simply a waste of time.

So, for avoiding all such backtracking and extra comparison we need to find prefix. ~~Pattern~~ occurance Pattern that are repeated can be tracked and reduce the waste of time.

eg. Pattern: a b c d a b c

prefix : ab, abc, abcd

suffix : c, bc, abc, dabc

→ In KMP, for a pattern we generate a π table.

→ π also known as Longest prefix (LPS)/suffix.

eg① P_1 = a b c d a b c d f ← Calculation of prefix π table generation

1 2 3 4 5 6 7 8 9

0 0 0 0 1 2 0 1 2 0

i	1	2	3	4	5	6	7	8	9	10	11
P_2	a	b	c	d	e	a	b	f	a	b	c
π	0	0	0	0	0	1	2	0	1	2	3

i	1	2	3	4	5	6	7	8	9	10
P_3	a	a	b	c	a	d	a	a	b	e
π	0	1	0	0	1	0	1	2	0	0

i	1	2	3	4	5	6	7	8	9
P_4	a	a	a	a	b	a	a	c	d
π	0	1	2	3	0	1	2	0	0

Example 1 KMP

String: a b a b c a b c a b a b a b d

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

j → 0

i	1	2	3	4	5
Pattern(i)	a	b	a	b	d
$\pi(i)$	0	0	1	2	0

Step 1: Construct a π-table as above.

Step 2. Point i to String[1] and j to Pattern[0]

Step 3. Match String[i] with ~~Pattern[j+1]~~ Pattern[j+1]

→ if they are same move i by 1 shift towards Right and j by 1 shift towards right

ie.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
String	a	b	a	b	c	a	b	c	a	b	a	b	a	b	d

j+1

i = 1 and j = 0

String[i] = Pattern[j+1]

i++ and j++

i	1	2	3	4	
P(i)	a	b	a	b	d
$\pi(i)$	0	0	1	2	0

Step 4 Again repeat the step - 3. until match is found

Here string[i] = pattern[j+1]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
String	a	b	a	b	c	a	b	c	a	b	a	b	a	b	d

i	1	2	3	4	5
patta(i)	a	b	a	b	d
π	0	0	1	2	0

increment i++, j++

→ Again repeat the step 3

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
String	a	b	a	b	c	a	b	c	a	b	a	b	a	b	d

i	1	2	3	4	5
P(i)	a	b	a	b	d
π	0	0	1	2	0

increment i++, j++

as string[i] = pattern[j+1]

→ Again repeat step 3.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
String	a	b	a	b	c	a	b	c	a	b	a	b	a	d	b

i	1	2	3	4	5
P(i)	a	b	a	b	d
π	0	0	1	2	0

increment i++, j++

as string[i] = Pattern[j+1]

→ Again repeat step - 3

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	a	b	a	b	c	a	b	c	a	b	a	b	a	d	b

i	1	2	3	4	5
P(i)	a	b	a	b	d
π	0	0	1	2	0

No match.

step 5 : If string [i] ≠ Pattern [j+1], move j to index π(i) pointed by (j+1) and check for step 4.

∴

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
String:	a	b	a	b	c	a	b	c	a	b	a	b	a	b	d

i

j

i	1	2	3	4	5
P(i)	a	b	a	b	d
π(i)	0	0	1	2	0

if still checking step 4 and there is no match, again move j to index π(i)

j

i	1	2	3	4	5
P(i)	a	b	a	b	d
π(i)	0	0	1	2	0

Again checking step 4, still no match, Now we cannot move j further left, so move/shift i by 1 in text string to right.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
String:	a	b	a	b	c	a	b	c	a	b	a	b	a	b	d

→ Again Repeat ~~from~~ step 3.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
String:	a	b	a	b	c	a	b	c	a	b	a	b	a	b	d

i	0	1	2	3	4	5
Pattern(i)		a	b	a	b	d
π(i)		0	0	1	2	0

Here i = 6 and j = 0, j+1 = 1.
∴ comparing string[i] with Pattern[j+1]

→ String[6] = P[1] ∴ i++, j++ ~~Match~~ string[8] ≠ P[3]
string[7] = P[2] ∴ i++, j++ ∴ j shifted to index of π(i) i.e. index-1

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
String:	a	b	a	b	c	a	b	c	a	b	a	b	a	b	d

$j \to 0$

i	1	2	3	4	5
P(i)	a	b	a	b	d
π	0	0	1	2	0

Since String[8] = P[1], shift j to 0 and since no further match is possible with P[j+1], move i by 1 towards Right.

→ Again Repeat step 3 until match is obtained.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
String	a	b	a	b	c	a	b	c	a	b	a	b	a	b	d

	0	1	2	3	4	5
i		1	2	3	4	5
P(i)		a	b	a	b	d
π(i)		0	0	1	2	0

String[9] = Pattern[j+1] ⇒ i++, j++

String[10] = Pattern[j+1] ⇒ i++, j++

String[11] = pattern[j+1] ⇒ i++, j++

String[12] = Pattern[j+1] ⇒ i++, j++

String[13] ≠ Pattern[j+1] ⇒ j shifted to index of π(P) ⇒ index = 2

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
String	a	b	a	b	c	a	b	c	a	b	a	b	a	b	d	

	0	1	2	3	4	5
i		1	2	3	4	5
P(i)		a	b	a	b	d
π(i)		0	0	1	2	0

String[13] = Pattern[j+1] ⇒ i++, j++

String[14] = Pattern[j+1] ⇒ i++, j++

String[15] = Pattern[j+1] ⇒ i++, j++

Since entire pattern is match, a b a b d is part of the text string.

* Complexity Analysis:

→ Function π is computed in O(m) time where as actual string matching routine scans n characters. So, worst case running time of algorithm is given by O(m+n).

Unit:- 5

Backtracking, Branch and Bound, NP Completeness:-

* Backtracking:

The basic idea of backtracking is to build up a vector, one component at a time and to test whether the vector formed has any chance of success.

→ The major advantage of this method is that we can realize the fact that the partial vector generated does not lead to an optimal solution

→ In such situation ~~than~~ be ignored.

→ Backtracking algorithm determines the solution by systematically searching the set of all feasible solutions for the given problem.

→ Follows (DFS)

* N- Queen Problem:

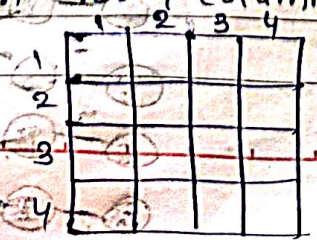
→ N- Queen problem is the classical example of backtracking.

→ N- Queen problem is defined as " given N×N chess board, arrange N queens in such a way that no two queens attack each other by being in same row, column or diagonal."

→ ~~So~~ For N = 1, this is a trivial case. For N = 2 and N = 3, solution is not possible, so we start with N = 4 and we will generalize it for N Queen.

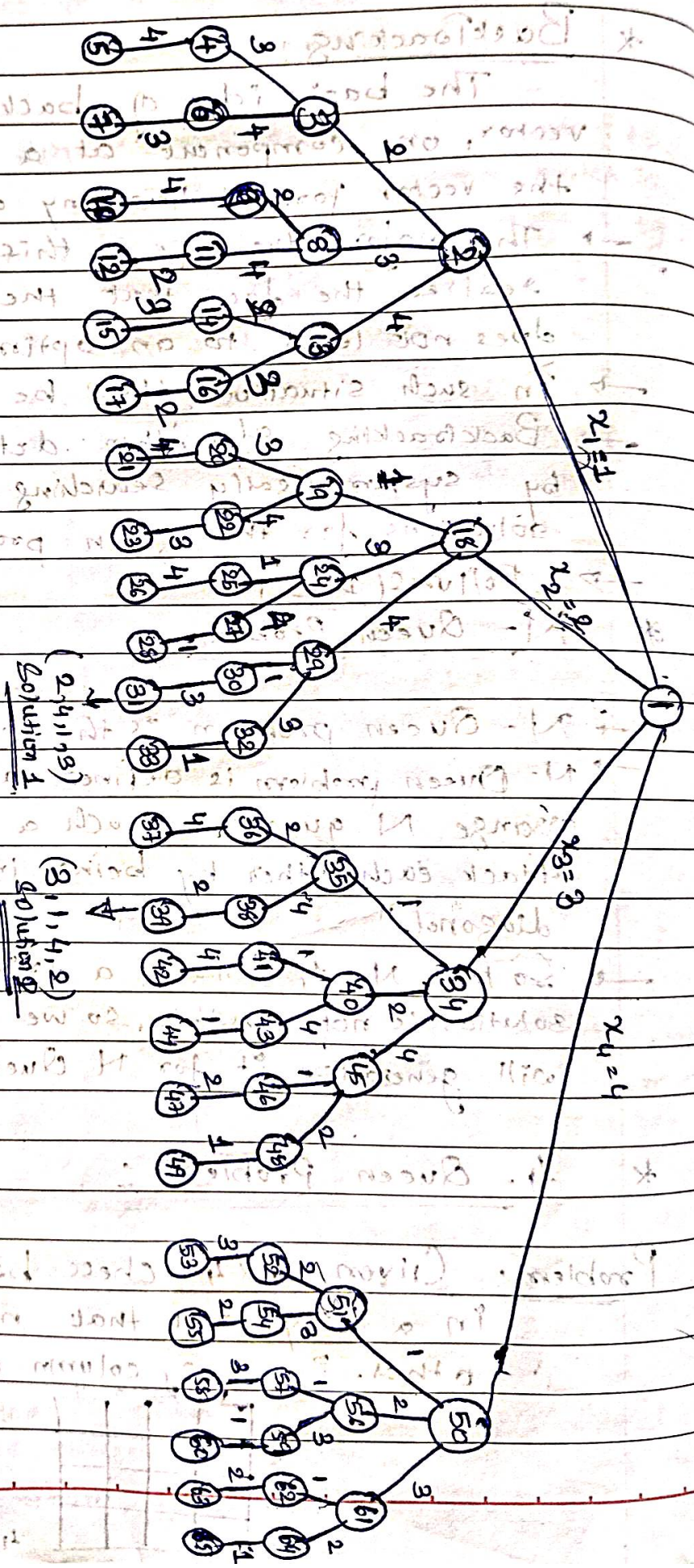
* 4- Queen Problem:

Problem: Given 4 × 4 chess board, arrange for queens in a way such that no two queen attack each other in row, column or diagonal.



4×4 Chessboard

Lets create a state space diagram for 4-queen problem.



State Space diagram for 4-queen problem

Here we have assumed Q_i queen will be placed only in i^{th} row.

Initially For 4-Queen Problem

	1	2	3	4
1	Q_1			
2			Q_2	
3				
4				

Q_3 queen cannot be placed on any column in row 3rd, so back tracking to Q_2.

	1	2	3	4
1	Q_1			
2				Q_2
3		Q_3		
4				

Q_4 cannot be placed on any column, so backtracking to Q_3, then back tracking to Q_2, then backtracking to Q_1.

	1	2	3	4
1		Q_1		
2				Q_2
3	Q_3			
4			Q_4	

Desired solution

Solution Vector

x_1	x_2	x_3	x_4
2	4	1	3

- Depth First Search technique is used.
- One problem has multiple solution.
- Solving by using backtracking reduces to check less possibilities.

⇒ Now we will consider how to place 8 Queens

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

Now, we will start placing the queens on the chessboard.

	1	2	3	4	5	6	7	8
1		Q_1						
2				Q_2				
3	Q_3							
4			Q_4					
5					Q_5			
6	x	x	x	x	x	x	x	x
7								
8								

→ Now, if we try to place Q_6 then it will not be possible to put it in a chessboard, so we need to back track

→ Similarly we check for each and every possibility and finally we obtain solution.

	1	2	3	4	5	6	7	8
1			Q_1					
2						Q_2		
3		Q_3						
4							Q_4	
5	Q_5							
6				Q_6				
7								Q_7
8					Q_8			

→ Final solution is $\{3, 6, 2, 7, 1, 4, 8, 5\}$

* Branch and Bound

→ Branch and bound is the systematic way for solving optimization problem. It is a counter part of backtracking. Backtracking explores the partial solution in DFS order, without any cost criteria. New solution is kept on stack.

→ B & B technique select solution having least cost criteria solutions. B & B technique visits explored node in BFS order, partial solutions are inserted in queue. In the next insertion, partial sol^n with minimum cost criteria is selected to expand it further.

→ B & B is applicable where greedy or dynamic programming fails. This approach is slower and in worst case it may lead to exponential running time.

* Knapsack Problem

→ is defined as " given set of items $I_1, I_2, \ldots, I_n$ having weight $w_1, w_2, \ldots, w_n$ and value $v_1, v_2, \ldots, v_n$, fill the knapsack of capacity W such that total weight of items in knapsack should not exceed the knapsack capacity and same time maximize the profit."

→ State space tree in case of knapsack would be a binary tree using Branch and Bound Technique. left edge indicate inclusion of items and right edge indicates exclusion of item.

→ Node structure is as follows:-

weight	value
Upperbound.	
Lchild	Rchild

→ All items are arranged in decreasing order of v_{i+1}/w_{i+1}

Upper bound is calculated as,

$$ub = v + (W - w) * (v_{i+1}/w_{i+1})$$

where, V → value associated with current item, W → capacity of knapsack.
w → weight of item under consideration.

Eg ① Find an optimal solution to the knapsack instance $n=4$, $M=8$, $(P_1, P_2, P_3, P_4) = (3, 5, 6, 10)$ and $(w_1, w_2, w_3, w_4) = (2, 3, 4, 5)$ using ~~backtracking~~ branch & bound. Also draw the corresponding state space tree.

Sol^n ⟶

Items	I_1	I_2	I_3	I_4
w_i	2	3	4	5
v_i	3	5	6	10
v_i/w_i	1.5	1.67	1.5	2

⟶ Now arrange it into descending order of v_i/w_i.

Items	I_4	I_2	I_1	I_3
w_i	5	3	2	4
v_i	10	5	3	6
v_i/w_i	2	1.67	1.5	1.5

Now, this is a maximization problem so let us compute upper bound for root node. Initially none of the item is selected, so.

$$ub = V + (W - w) * (v_{i+1}/w_{i+1})$$

It: 0

$$\therefore ub = 0 + (8 - 0) * 2$$
$$= 16$$

Initial state space tree:

W=0	V=0
ub = 16	

It. 1:

Now, we have two possibilities. Include I_4 or exclude I_4

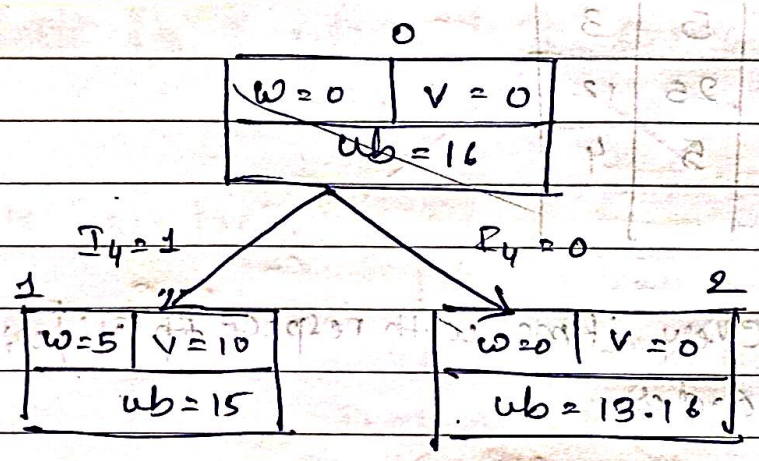
Include I_4:

$ub = 10 + (8-5) * 1.67 = 10 + 5 = 15$

Exclude I_4

$ub = 0 + (8-0) * 1.67 = 13.36$

State Space Tree:



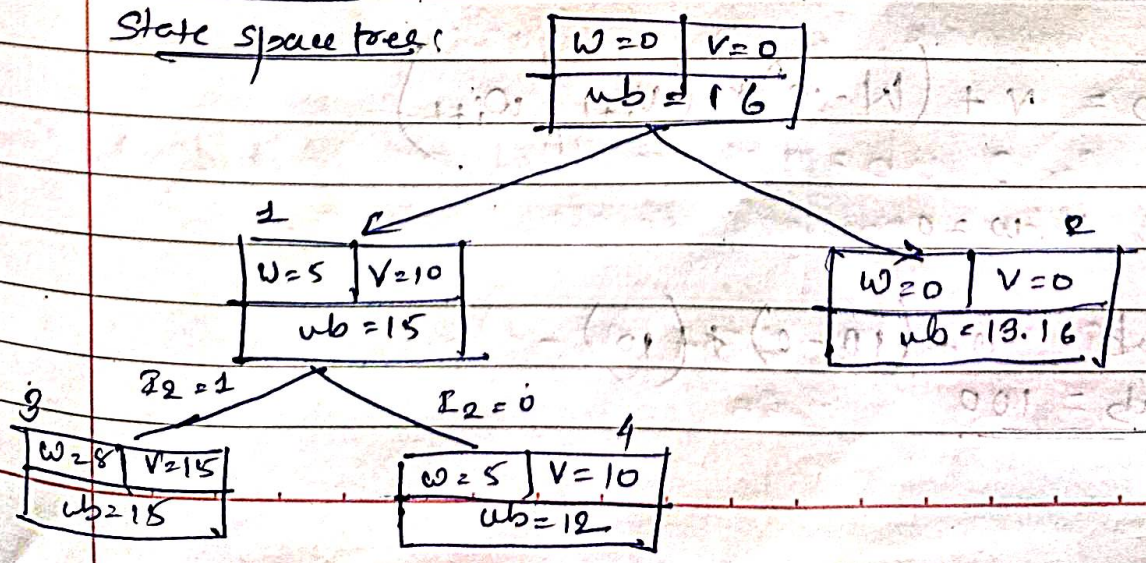
It 2 Again two choices, Include I_2 or exclude I_2

Include I_2:

$ub = 15 + (8-8) * 1.5 = 15 + 0 = 15$

Exclude I_2: $ub = 0 + (8-0) * 1.5 = 12$

State space tree:



→ Knapsack is full so stop. Solution obtained by branch and bound technique selects items $\langle I_4, I_2 \rangle$ which gives profit of 15.

Example (2) Consider set of items $I = \langle I_1, I_2, I_3, I_4 \rangle$ having weight $w = \{4, 7, 5, 3\}$ and values $v = \langle 40, 42, 25, 12 \rangle$. Solve the knapsack problem using branch and bound with knapsack capacity $W = 10$

Sol^n ⟹

Items	I_1	I_2	I_3	I_4
w_i	4	7	5	3
v_i	40	42	25	12
v_i/w_i	10	6	5	4

Now, arrange every items with respect to v_i/w_i into its decreasing order.

Items	I_1	I_2	I_3	I_4
w_i	4	7	5	3
v_i	40	42	25	12
v_i/w_i	10	6	5	4

Iteration 0.

$$ub = v + (W - w) * (v_{i+1} / w_{i+1})$$

Initially $v, w = 0$

$$\therefore \quad ub = 0 + (10 - 0) * (10)$$

$$ub = 100$$

∴ State space tree:

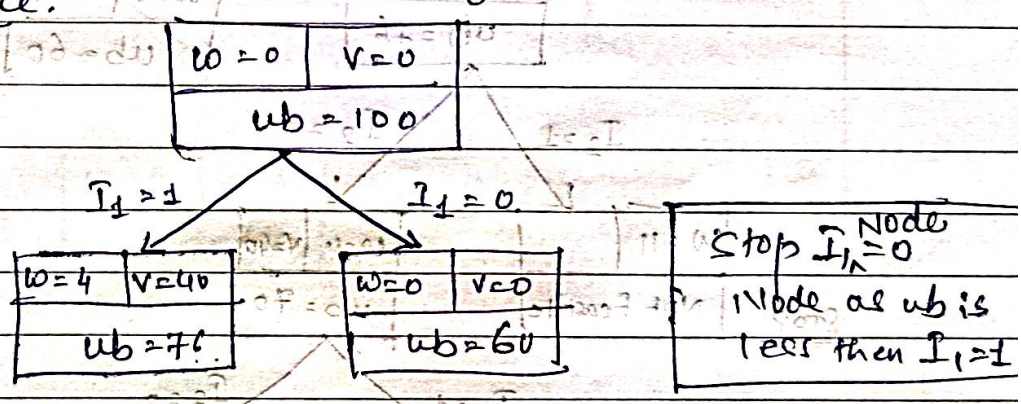
W=0	V=0
ub = 100	

Step 2:- Either we should include I_1 or Exclude I_1

∴ Include I_1 : $ub = 40 + (10-4) * 6$
$= 40 + 36$
$= 76$

∴ ~~Exclude~~ Include I_2 : $ub = 0 + (10-0) * 6$
$= 60$

∴ State space tree:

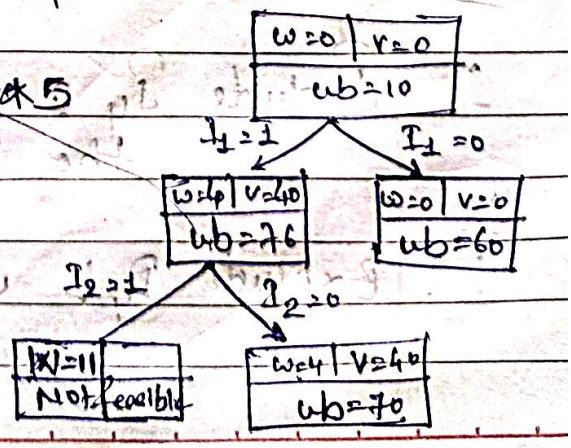


Step 3: Either we should include I_2 or Exclude I_2

∴ Include I_2 : Inclusion of I_1 and I_2 both exceed the knapsack capacity, so prune this branch.

∴ Exclude I_2 : $ub = 40 + (10-4) * 5$
$= 70$

∴ State Space Tree:

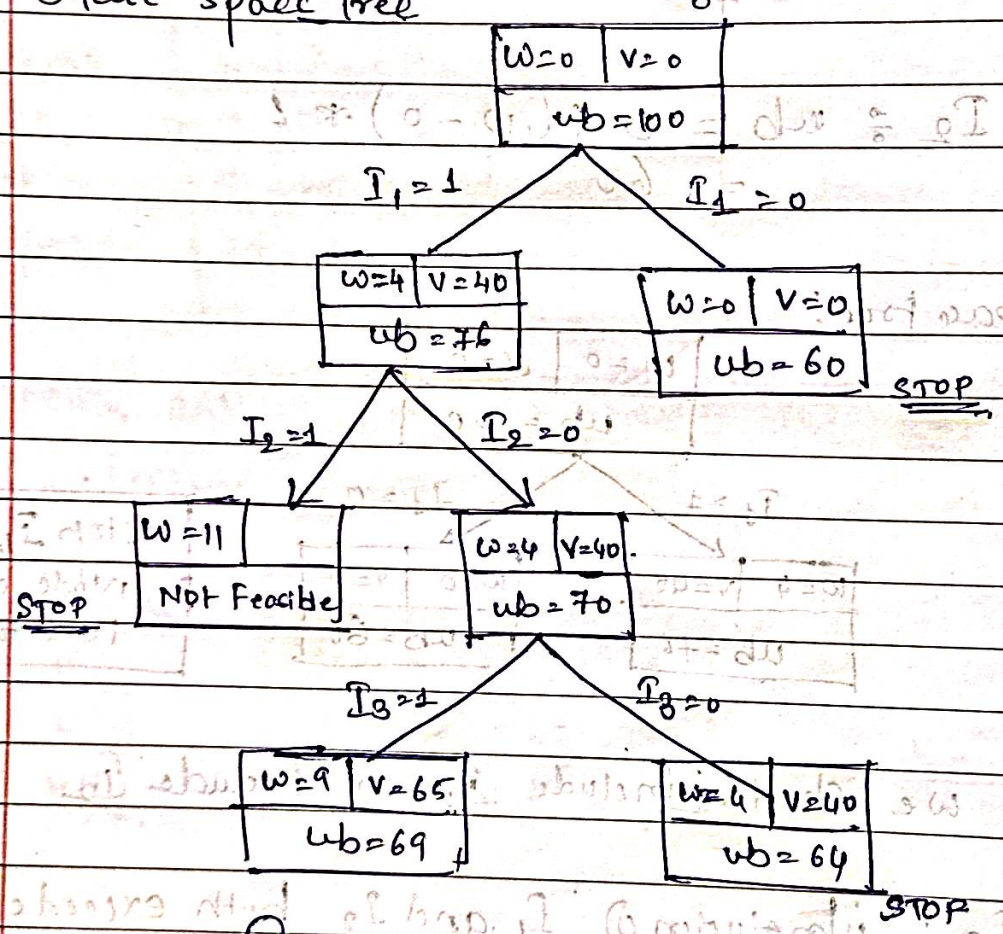


It 4. Either Include I_3 or Exclude I_3

∴ Include I_3 : $ub = 75 + (10-9) * 4$
$= 69$

∴ Exclude I_3 : $ub = 40 + (10-4) * 4$
$= 64$

∴ State Space Tree

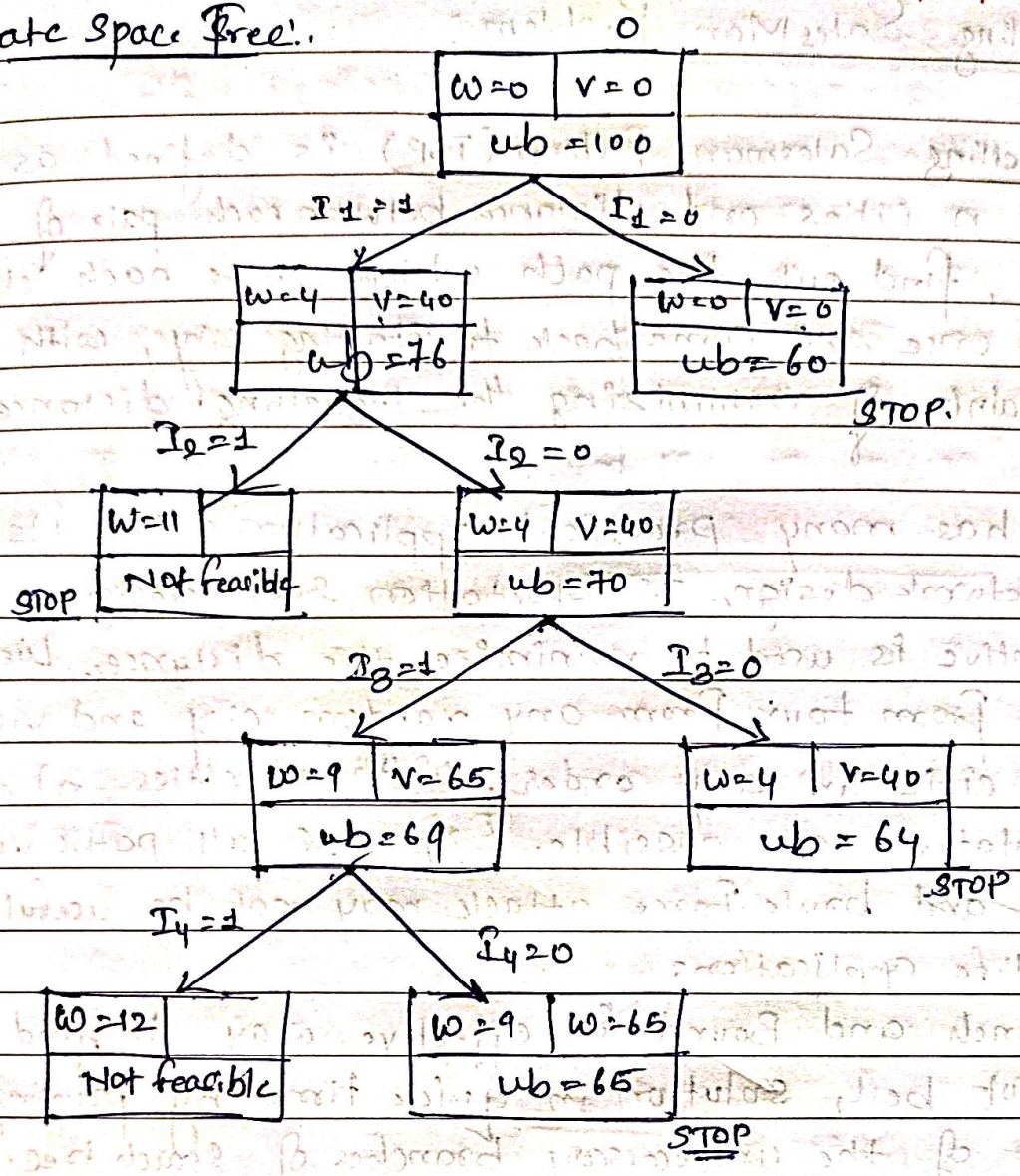


It 5 Either Include I_4 or Exclude I_4

∴ Include I_4 : ub = Not feasible as Including I_4 exceeds the knapsack weight capacity.

∴ Exclude I_4 : $ub = 65 + (10-9) * 0$
$= 65$

∴ State Space Tree:



∴ Solution obtained by branch and bound technique; selects items $\langle I_1, I_2 \rangle$ which gives profit of 65.

* Travelling Sales Man Problem :-

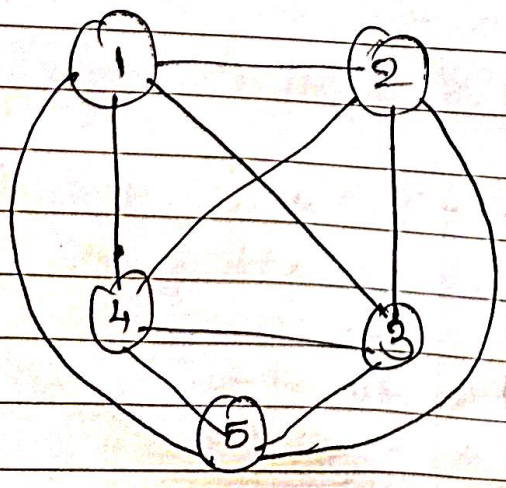
→ Travelling Salesman Problem (TSP) is defined as "given n cities and distance between each pair of cities, find out the path which visits each city exactly once and come back to starting city, with constraint of minimizing the travelling distance.

→ TSP has many practical application's like its used in network design, transportation route design.

→ Objective is used to minimize the distance. We can start ~~from~~ tour from any random city and visit other cities in any order. With n cities, n! different permutations are possible. Exploring all paths using ~~branch and~~ brute force attack may not be useful in real life applications.

→ Branch and Bound is effective way to find better if not best, solution in quick time by pruning some of the unnecessary branches of search tree.

Example. Solve the following TSP using Branch and Bound technique with starting node 1.



Adjacency Matrix

	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	∞

DOMS Page No.
Date / /

→ Step 1: Reduce the given adjacency Matrix.

(1) → First take minimum value in every row

(2) → Then, subtract that minimum value from every element of row. as follows

(1)

	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

→ Select Minimum value in every row

(2)

1	∞	10	20	0	11	10
2	13	∞	14	2	0	2
3	1	3	∞	0	2	2
4	16	3	15	∞	0	3
5	12	0	3	12	∞	4
						21

→ Subtracted minimum value from each row.

→ Step 2:

(3) → Second Take minimum value in each column.

(4) → Then, subtract that minimum value from that column as follows.

(3)

∞	10	20	0	11	10
13	∞	14	2	0	2
1	3	∞	0	2	2
16	3	15	∞	0	3
12	0	3	12	∞	4
					21
1	0	8	0	0	

→ Selection of minimum value in each column

(4)

∞	10	17	0	11	10
12	∞	11	2	0	2
0	3	∞	0	2	2
15	3	12	∞	0	3
11	0	0	12	∞	4
					21
1	0	3	0	0	

∴ Reduced Matrix and Reduced cost is 25

ORIGINAL MATRIX (ii)

	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	∞

Reduced Matrix

Reduced cost = 25

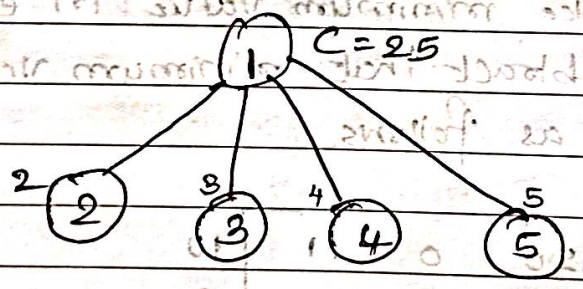
Step 2: Now, solve the Problem using Branch and Bound.

→ Therefore we have to draw state space tree with upper bound value (i.e. cost of 1st node = 25)

→ Here cost of Initial upper/upperbound value to ∞ and update when we reach the leaf node

→ For each node we have to calculate the cost/~~reduced cost~~.

upper = ∞



Now. * Cost of node 2 :-

(ORIGINAL Adjacency Matrix)

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

Steps → Make 1st row and 2nd column value as ∞.

→ Check if it can be further reduced by checking all row or column has atleast 0 or all ∞ values.

→ We should not go back to 1 so, 2 to 1 is set to ∞.

Following above steps, adjacency matrix is as follows.

(03 1.3)

$$\begin{array}{c} \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 & 0 & 0 \end{matrix} \end{array}$$

→ Check if it can be reduced

Cond 1 → Check if entire row is ∞ or any value in row is 0. then final value set to 0.

Cond 2 → Check if entire column is ∞ or any one value in column is zero(0) then set final column value to 0.

This adjacency matrix is already reduced

Note: If both conditions in which any column is not full ∞ or not any zero value, then reduce the adjacency matrix.

∴ The cost of node 2 $\boxed{c(2)}$ $= c(1,2) + c(1) + \hat{r}$

$= 10 + 25 + 0$

$= 35$

Note: $c(1,2)$ → cost from 1 to 2 in Original matrix (1)

i.e. $c(1,2) = 10$

$c(1)$ → Reduced cost at previous node

i.e. Cost at node 1, $c(1) = 25$

$\hat{r}$ → Reduction in the current matrix.

i.e. 0

* Cost of Node 3:-

:→ Again taking the original Matrix (i1) and performing the steps as performed for calculation of cost of nodes as follows:- $\boxed{1^{st} \text{ row as } \infty \text{ and } 3^{rd} \text{ column to } \infty \text{ and } C(3,1) = \infty}$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty & 0 \\ 12 & \infty & \infty & 2 & 0 & 0 \\ \infty & 3 & \infty & 0 & 2 & 0 \\ 15 & 3 & \infty & \infty & 0 & 0 \\ 11 & 0 & \infty & 12 & \infty & 0 \\ 11 & 0 & 0 & 0 & 0 & \hat{r} = 11 \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty & 0 \\ 1 & \infty & \infty & 2 & 0 & 0 \\ \infty & 3 & \infty & 0 & 2 & 0 \\ 4 & 3 & \infty & \infty & 0 & 0 \\ 0 & 0 & \infty & 12 & \infty & 0 \\ 0 & 0 & 0 & 0 & 0 & \hat{r} = 11 \end{bmatrix}$$

This can be further reduced as minimal value can be obtained except '0'.

⇒ The above adjacency matrix is reduced.

$$\therefore C(3) = C(1,3) + C(1) + \hat{r}$$
$$= 17 + 25 + 11$$
$$= \underline{\underline{53}}$$

* Cost for Node 4:-

⇒ Following the same steps done for node 2 & 3, find for node 4.

→ Put ∞ in 1st row and 4th column & value at $C(4,1)$ as ∞.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty & 0 \\ 12 & \infty & 11 & \infty & 0 & 0 \\ 0 & 3 & \infty & \infty & 2 & 0 \\ \infty & 3 & 12 & \infty & 0 & 0 \\ 11 & 0 & 0 & \infty & \infty & 0 \\ 0 & 0 & 0 & 0 & 0 & \end{bmatrix}$$

⇒ No further reduction required. $\therefore \hat{r} = 0$

$$\therefore Cost(4) = C(1,4) + C(1) + \hat{r}$$
$$= 0 + 25 + 0$$
$$= \underline{\underline{25}}$$

* Cost at Node 5:-

→ Putting ∞ in 1st row as 5th column and value of c(5,1) as ∞. in original matrix (i1) as follows.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 17 & 2 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 12 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix} \begin{matrix} 0 \\ 2 \\ 0 \\ 3 \\ 0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 12 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

$$\begin{matrix} 5 & & & & & \sum 5 = \hat{r} \\ & 0 & 0 & 0 & 0 & \end{matrix}$$

Reduction of matrix in row.

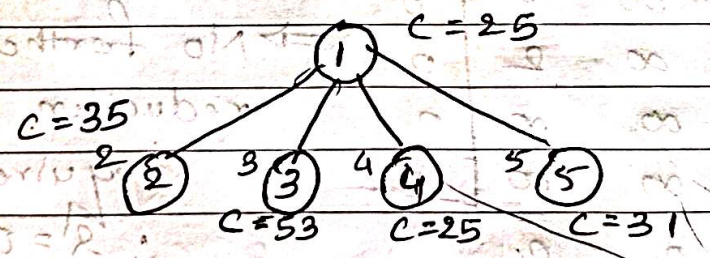
→ No reduction required in column.

$\therefore C(5) = C(1,5) + C(1) + \hat{r}$

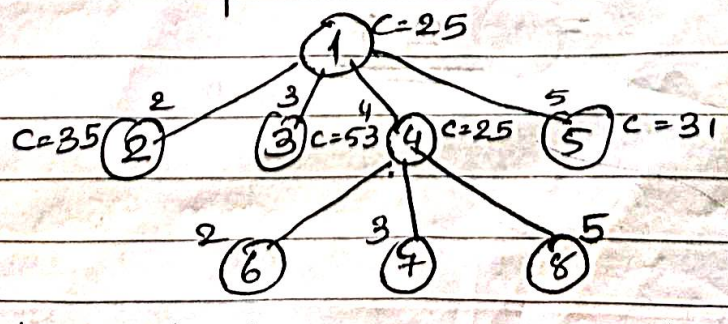
$= 1 + 25 + 5$

$= \underline{\underline{31}}$.

∴ State space tree is as follows:-



→ Over here, path 1 → 4 is minimum, so we will explore the path further from 1 → 4.



→ Exploring node 4, we can go to 4 → 1, 4 → 3, 4 → 5, and find cost at each node.

* Cost for Node 6 : (Path 4 → 2)

→ To calculate the cost for node 6, we have to take original reduced matrix of node 4 i.e. (i.e.)

$$\begin{array}{c|ccccc} & & & & & \\ \hline & \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{array}$$

→ For Path (4 → 2), put 4 row values as ∞ and 2nd column value as ∞. Also path 4 → 2 is considered as path (1 → 4 → 2) i.e. value at C(2, 1) as also ∞. in above adjacency Matrix.

$$\begin{array}{c|ccccc|c} & 1 & 2 & 3 & 4 & 5 & \\ \hline 1 & \infty & \infty & \infty & \infty & \infty & 0 \\ 2 & \infty & \infty & 11 & \infty & 0 & 0 \\ 3 & 0 & \infty & \infty & \infty & 2 & 0 \\ 4 & \infty & \infty & \infty & \infty & \infty & 0 \\ 5 & 11 & \infty & 0 & \infty & \infty & 0 \\ & 0 & 0 & 0 & 0 & 0 & \end{array}$$

⇒ No further reduction required

$\therefore \hat{r} = 0$

$\therefore$ Cost(6) = Cost(4, 2) + c(4) + $\hat{r}$

= 3 + 25 + 0

= 28

* Cost at Node (7) :- (Path 4 → 3)

⇒ Following the same steps as followed for calculation of Node 6. with adjacency matrix reference (i2).

⇒ Path 4 → 3 ⇒ 1 → 4 → 3 ⇒ $c(3,1) = \infty$ and 1st row as ∞ and 3rd column as ∞ as follows:-

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & 3 & \infty & \infty & 2 \\ \infty & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 2 \\ 0 \\ 0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & \infty & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

$$\begin{matrix} 11 & 0 & 0 & 0 & 0 \end{matrix} \quad 0 + = 13 \qquad \begin{matrix} 0 & 0 & 0 & 0 & 0 \end{matrix} \quad 13 = \hat{r}$$

Unreduced Matrix — Reduced Adj. Matrix

$$\therefore \text{Cost } c(7) = c(4,3) + c(4) + \hat{r}$$
$$= 12 + 25 + 13$$
$$= \underline{\underline{50}}$$

* Cost at Node (8) : (Path 4 → 5)

⇒ Path 4 → 5 ⇒ 1 - 4 - 5 ⇒ $c(1,5) = \infty$ and put 4th row and 5th column value as ∞. in Matrix (i2).

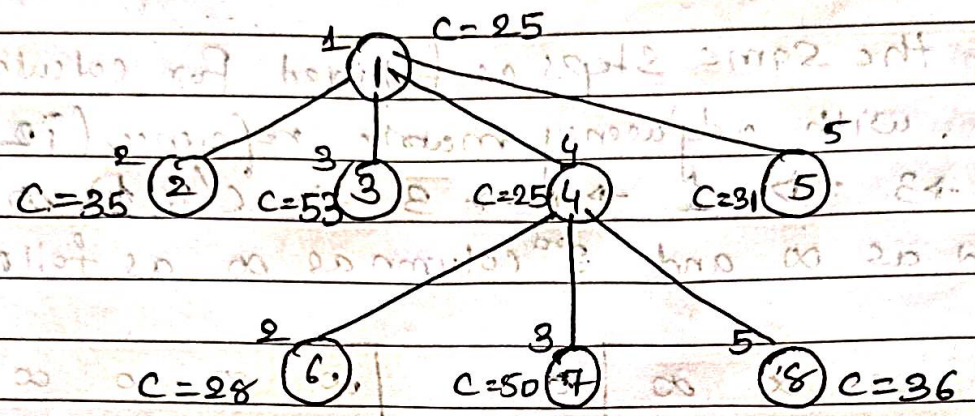
$$\therefore \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix} \begin{matrix} 0 \\ 11 \\ 0 \\ 0 \\ 0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \end{matrix} \quad 11 \qquad \begin{matrix} 0 & 0 & 0 & 0 & 0 \end{matrix} \quad 11 = \hat{r}$$

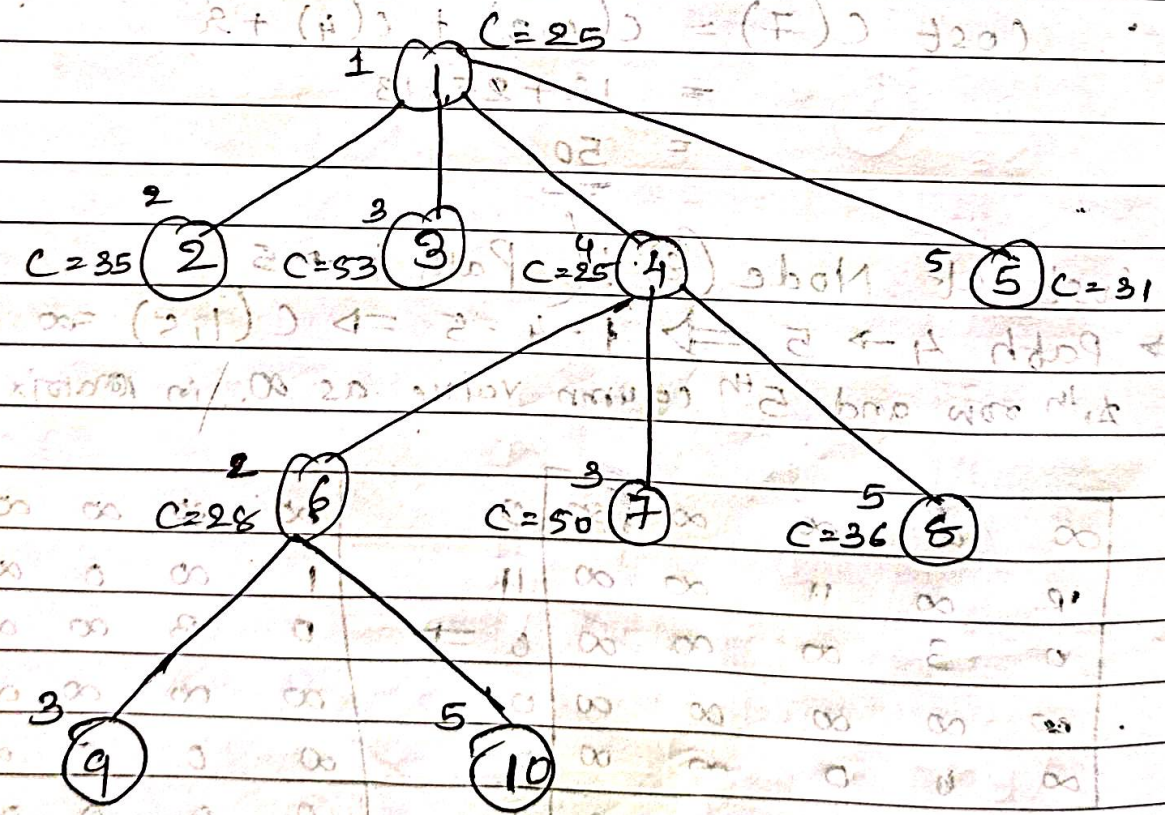
Unreduced matrix

$$\therefore c(8) = c(4,5) + c(4) + \hat{r}$$
$$= 0 + 25 + 11$$
$$= \underline{\underline{36}}$$

∴ State Space Tree is as follows with cost-



⇒ Exploring path for node 4, we found out that path from (Path 4 → 2) is minimum with cost=25 so we will explore with the same path further as follows.



⇒ Now there are two path from ~~Node~~ (2 → 3) (9) and (2 → 5) (10). Find the cost at Node 9 and Node 10 further.

* Cost at Node 9 :- (Path 2 → 3)

Take (i3) as reference adjacency Matrix and proceed further

$$\begin{array}{|ccccc|c} \infty & \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & 2 & 0 \\ \infty & \infty & \infty & \infty & \infty & 0 \\ 11 & \infty & \infty & \infty & \infty & 11 \\ \hline & & & & & 11 \end{array} \Rightarrow \begin{array}{|ccccc|c} \infty & \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & 0 & 2 & 0 \\ \infty & \infty & \infty & \infty & \infty & 0 \\ 0 & \infty & \infty & \infty & \infty & 0 \\ \hline 0 & 0 & 0 & 0 & 2 & 11 \end{array}$$

Reduced row matrix

$$\begin{array}{|ccccc|c} \infty & \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & 0 & 0 & 0 \\ \infty & \infty & \infty & \infty & \infty & 0 \\ 0 & \infty & \infty & \infty & \infty & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 13 = \hat{r} \end{array}$$

Reduced column Matrix

$$\therefore c(q) = c(2,3) + c(6) + \hat{r}$$
$$= 11 + 28 + 13$$
$$= \underline{\underline{52}}$$

* Cost at Node 10 : (Path 2 → 5)

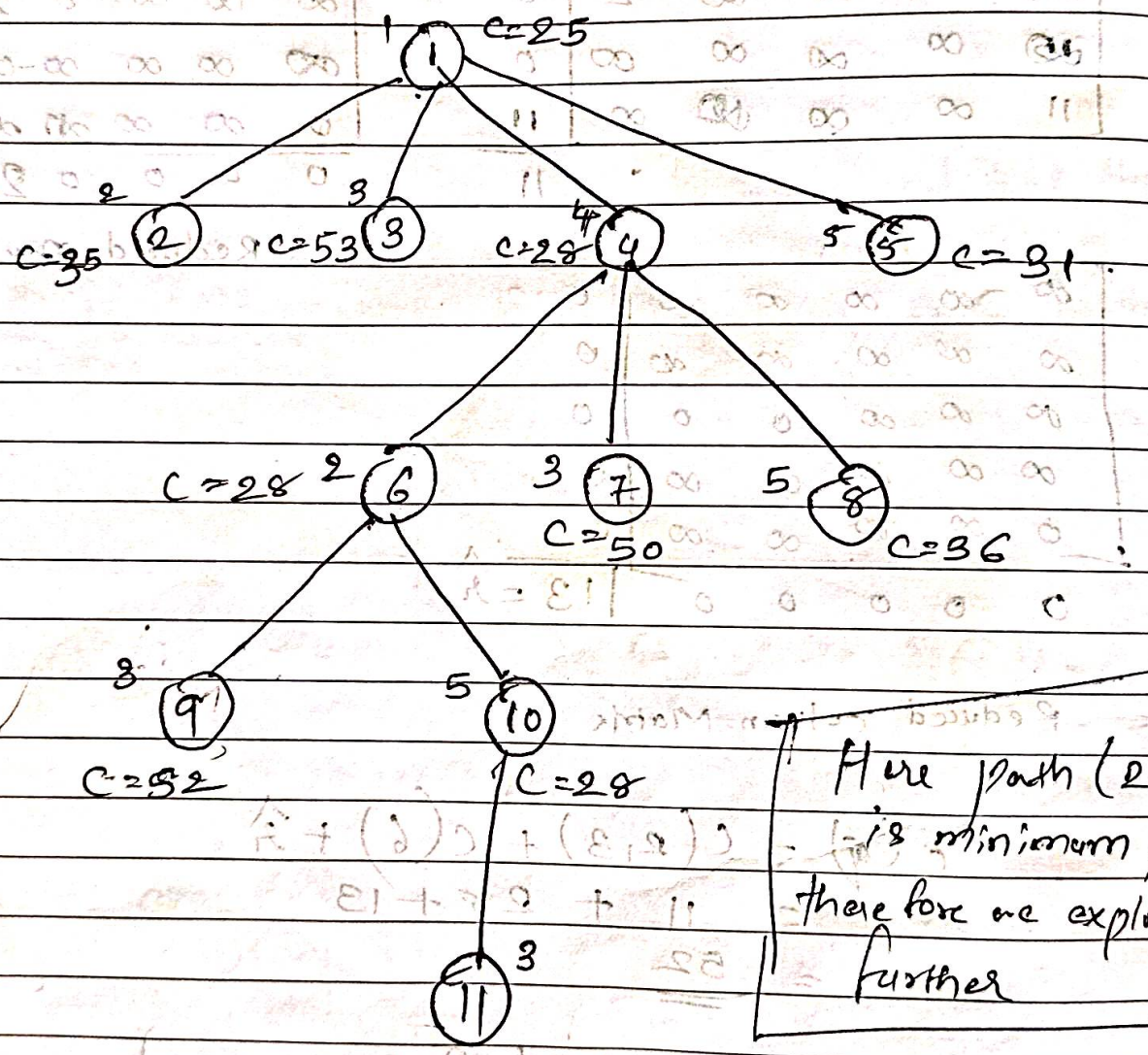
$$\begin{array}{|ccccc|c} \infty & \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty & 0 \\ 0 & \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & 0 & \infty & \infty & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & \end{array}$$

⇒ No reduction required further

$\therefore \hat{r} = 0$

$$\therefore C(10) = C(2,5) + C(6) + \hat{r}$$
$$= 0 + 28 + 0$$
$$= \underline{\underline{28}}$$

∴ State Space Tree is as follows:-



Here path (2 → 5) is minimum, therefore we explore further

* Cost at Node (11) (Path 5 → 3)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

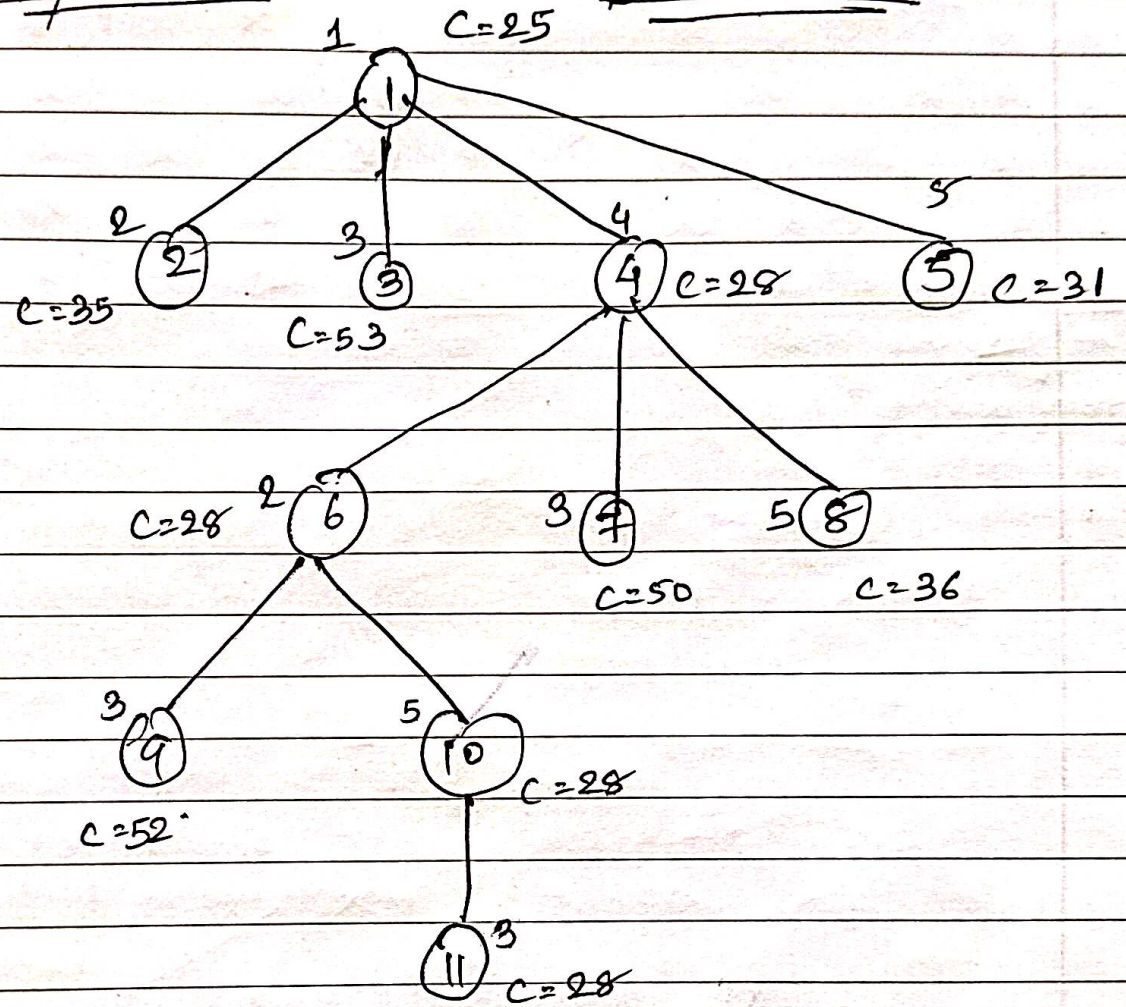
$\therefore c(11) = c(5,3) + c(6) + \hat{r}$

$= 0 + 28 + 0$

$= \underline{\underline{28}}$

$\boxed{\text{Upper} = 28}$

$\therefore$ State Space tree $\Rightarrow$



$\therefore$ All the nodes Cost is greater then upper, so, all the nodes will be killed, therefore no need to explore.

update Upper $= \infty$ to upper $= \underline{\underline{28}}$

$\therefore$ Shortest Path:

$\Rightarrow 1 \to 4 \to 2 \to 5 \to 3 \to 1$