MCM

MCM is matrix chain multiplication. Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications but merely to decide in which order the multiplications are to be performed.

```
Matrix-Chain-Order (p)
n ← length[p] - 1
for i = 1 to n
        do m[i,i] ← 0
for i = 2 to n
        do for i ← 1 to n - l + 1
                do j ← i + l - 1
                   m[i,j] ← ∞
                   for k ← i to j-1
                      do q ← m[i,k] + m[k+1,j] + p_{i-1} p_k p_j
                      if q < m[i,j]
                          then m[i,j] ← q
                               s[i,j] ← k
return m and s
```

LCS

Longet common subsequence

Given two sequences, find the length of longest subsequence present in both of them.

A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

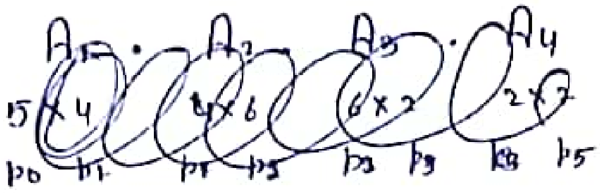
Eg - abcdefg ← sequence

subsequences → abc, abg, abd, aeg, ----

So a string of length n has 2^n different possible subsequences.

There are 2 methods of LCS
- recursion
- dynamic programming

Matrix Chain Multiplication U-2



$A_1 \cdot A_2 \cdot A_3 \cdot A_4$

A_1	A_2	A_3	A_4
5×4	4×6	6×2	2×7
p_0 p_1	p_1 p_2	p_2 p_3	p_3 p_4

$$M[i,j] = \begin{cases} 0, & i = j \\ \underset{i \le k < j}{\text{Min}}\ M[i,k] + M[k+1,j] + p_{i-1}\,p_k\,p_j \end{cases}$$

	1	2	3	4
1	0	120		
2	X	0	48	
3	X	X	0	84
4	X	X	X	0

$i = 1 \quad j = 2 \quad k = 1$

$M[1,2] = 0 + 0 + 5 \times 4 \times 6$
$= 120$

$i = 2 \quad j = 3 \quad k = 2$

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

$i = 3 \quad j = 4 \quad k = 3$

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

and so on .....

→ Verification of M[1, 4] at k = 3

M[1, 3] + M[4, 4]

$\to$ 0

↓

k = 1 M[1, 1] + M[2, 3]

0

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

4 × 6 × 2 — 48

4 × 2

5 × 4 × 2 — 40

5 × 2

5 × 2 × 7 — 70

158

Longest Common Subsequence LCS

⇒ Algo to find LCS using recursion.

```
int LCS (i, j)
{
    if (A[i] == '\0' || B[j] == '\0')
        return 0;
    else if (A[i] == B[j])
        return 1 + LCS (i+1, j+1);
    else
        return Max (LCS (i+1, j), LCS (i, j+1));
}
```

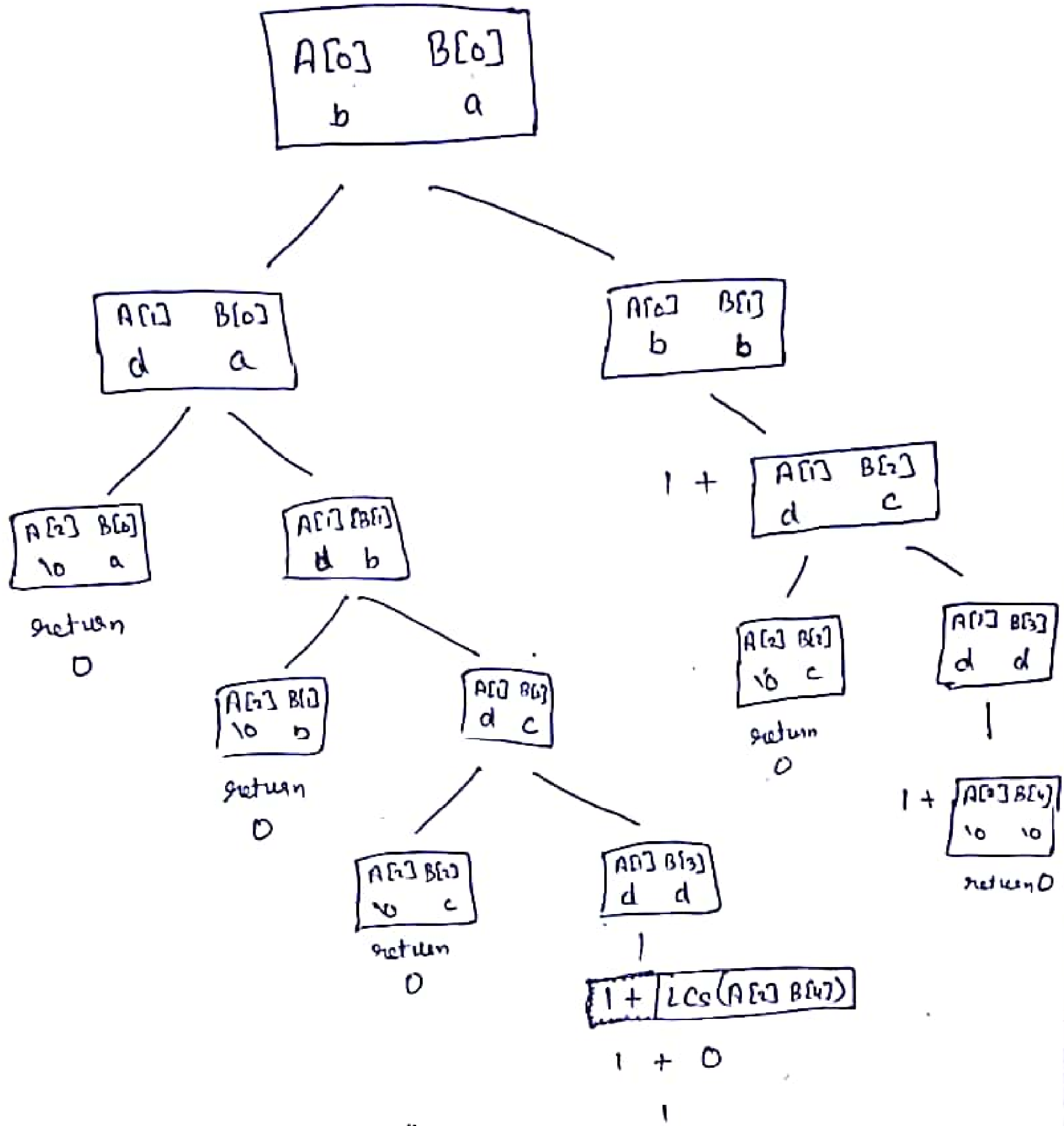
9 -

String 1

0	1	2
b	d	\0

String 2

0	1	2	3	4
a	b	c	d	\0



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

2

LCS = 2

⇒ Algo using dynamic programming

if (A[i] == B[j])

LCS [i, j] = 1 + LCS [i-1, j-1]

else

LCS [i, j] = Max [LCS (i-1, j), ~~[illegible]~~ LCS (i, j-1)]

Eg – String 1 | b | d |

String 2 | a | b | c | d |

		a	b	c	d
	0	1	2	3	4
0	0	0	0	0	0
b 1	0	0	1	1	1
d 2	0	0	1	1	2

DAA

Greedy Algorithm

Activity selection problem

- Sort the activity in ascending order of their finishing time.
- Select the first activity
- Select the new activity if its starting time is greater or equal to the finishing time of previously selected. activity.
- Repeat step 3 till all the activities are examined.

Algorithm -

```
n ← length [s]
A ← {1}
j ← 1
for i=2 to n
    do if si > fj
        then A ← A ∪ {i}
             j ← i
return A
```

Example - $S = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8\}$

$S_i = \{1, 2, 3, 4, 8, 9, 9, 11\}$

$f_i = \{3, 5, 4, 7, 10, 9, 11, 12\}$

Step 1 - Arrange them in increasing order of finishing time -

Activity	A_1	A_3	A_2	A_4	A_6	A_5	A_7	A_8
Start Time	1	3	2	4	9	8	9	11
Finish Time	3	4	5	7	9.	10	11	12

i, j → j, i

starting time of j ≥ finishing time of i

$1 \not\geq 3$ j shifts

$3 \geq 3$ i shifts

$3 \not\geq 4$ j shifts

and so on ---

selected activity	A_1	A_3	A_4	A_6	A_7	A_8
start	1	3	4	9	9	11
finish	3	4	7	9	11	12

Huffman code.

Data can be encoded efficiently using Huffman codes. It is widely used and very effective technique for compressing data.

Eg - B C C A B B D D A E C C B B A E D D C C

(i) Fixed length code.

A - 65 - 0100 0001 - 8 bit

B - 66 - 0100 0010

C - 67 - 0100 0011

D - 68 - 0100 0100

E - 69 - 0100 0101

Char	Count	code
A	3	000
B	5	001
C	6	010
D	4	011
E	2	100
	20	

msg size - 20 × 3 = 60

code size - 5 × 3 = 15

char - 5 × 8 = 40

115

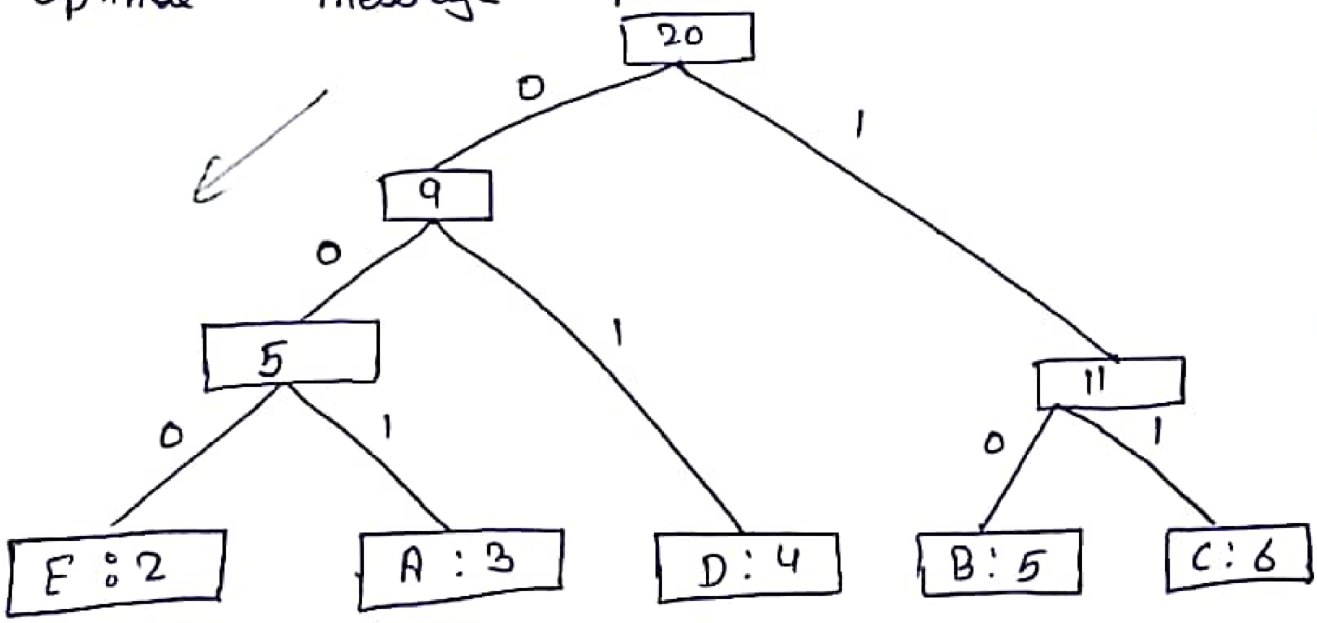
(iii) Variable length code

Char	Count	Code
A	3	~~1~~001
B	5	10
C	6	11
D	4	01
E	2	000

A → 3×3 – 9
B → 5×2 – 10
C – 12
D – 8
E – 6

45

* Optimal message pattern tree



msg size – 45
code size – 12
char – 40

97

Algorithm -

```
HUFFMAN (C)
n ← |C|
Q ← C
for i = 1 to n-1
        do allocate a new node z
        left[z] ← x ← Extract_Min (Q)
        right[z] ← y ← Extract_Min (Q)
        f[z] ← f[x] + f[y]
        Insert (Q, z)
return Extract_Min (Q)
```

Eg -

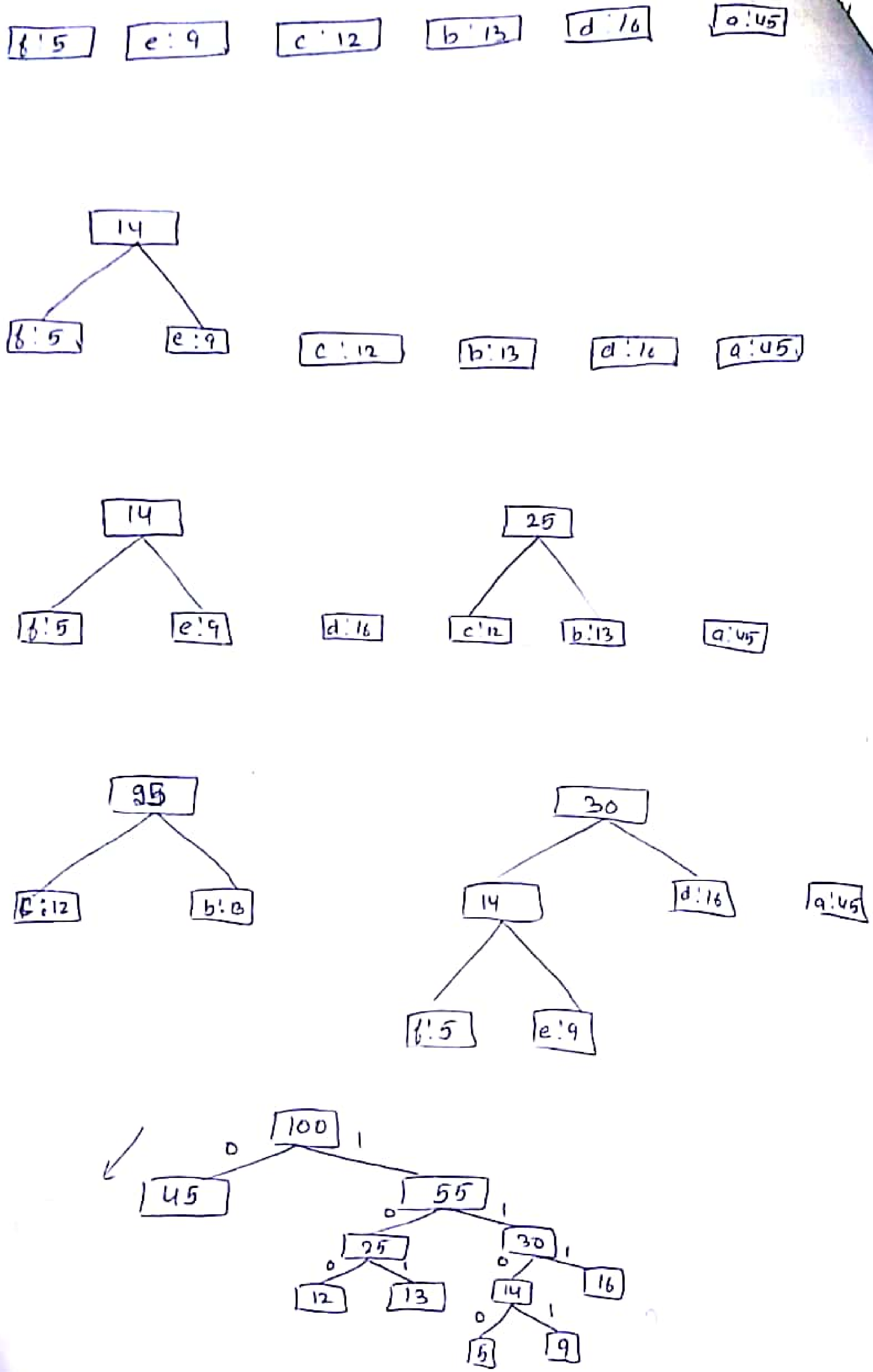
Char	Count	Code
a	45	0
b	13	101
c	12	100
d	16	111
e	9	1101
f	5	1100

a → 45 × 1 = 45
b → 39
c → 36
d → 48
e → 36
f → 20

224

msg size - 224
code size - 100
char - 40

364


f:5
e:9
c:12
b:13
d:16
a:45
14
25
30
100
45
55
12
13
16
5
9
0
1

Task Scheduling Problem

This is the problem of optimally scheduling unit time tasks on a single processor, where each task has a deadline and some profit.

Eg –

Task	T_1	T_2	T_3	T_4	T_5	T_6
Profit	50	40	30	10	20	1
Deadline	2	2	3	4	1	5

Arrange profit in descending order.

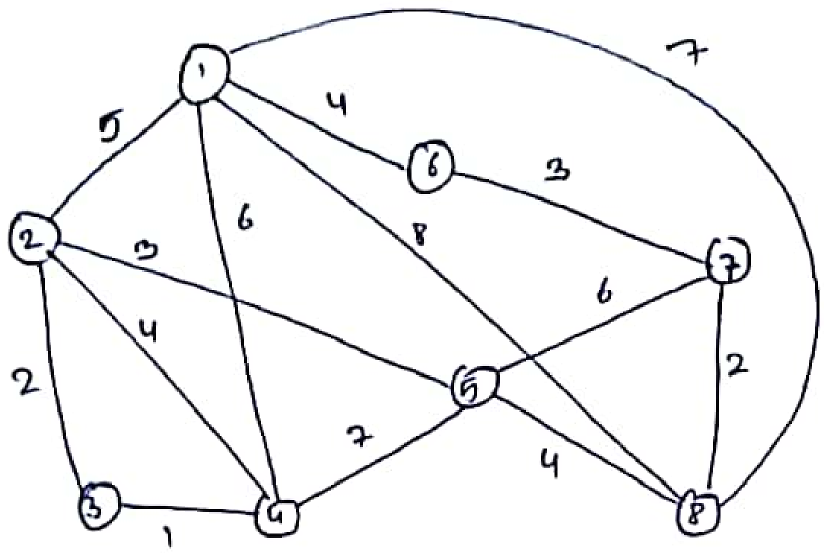
Task	T_1	T_2	T_3	T_5	T_4	T_6
Profil	50	40	30	20	10	1
Deadline	2	2	3	1	4	5

0–1	1–2	2–3	3–4	4–5	
T_2	T_1	T_3	T_4	T_6	
40	50	30	10	1	= 131

Travelling Salesman Problem

In this a salesman needs to visit 'n' cities in such a manner that all cities must be visited at once and in the end he returns to the city from where he started with minimum cost.

Eg -

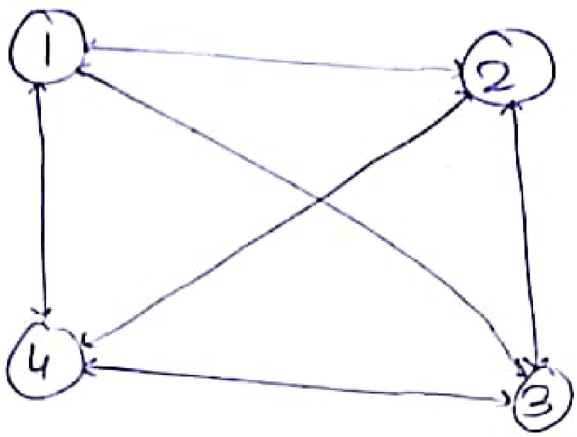


	1	2	3	4	5	6	7	8
1	0	5	0	6	0	4^{1}	0	7
2	5	0	2^{6}	4	3	0	0	0
3	0	2	0	1^{7}	0	0	0	0
4	6^{8}	4	1	0	7	0	0	0
5	0	3^{5}	0	7	0	0	6	4
6	4	0	0	0	0	0	3^{2}	0
7	0	0	0	0	6	3	0	2^{3}
8	7	0	0	0	4^{4}	0	2	0

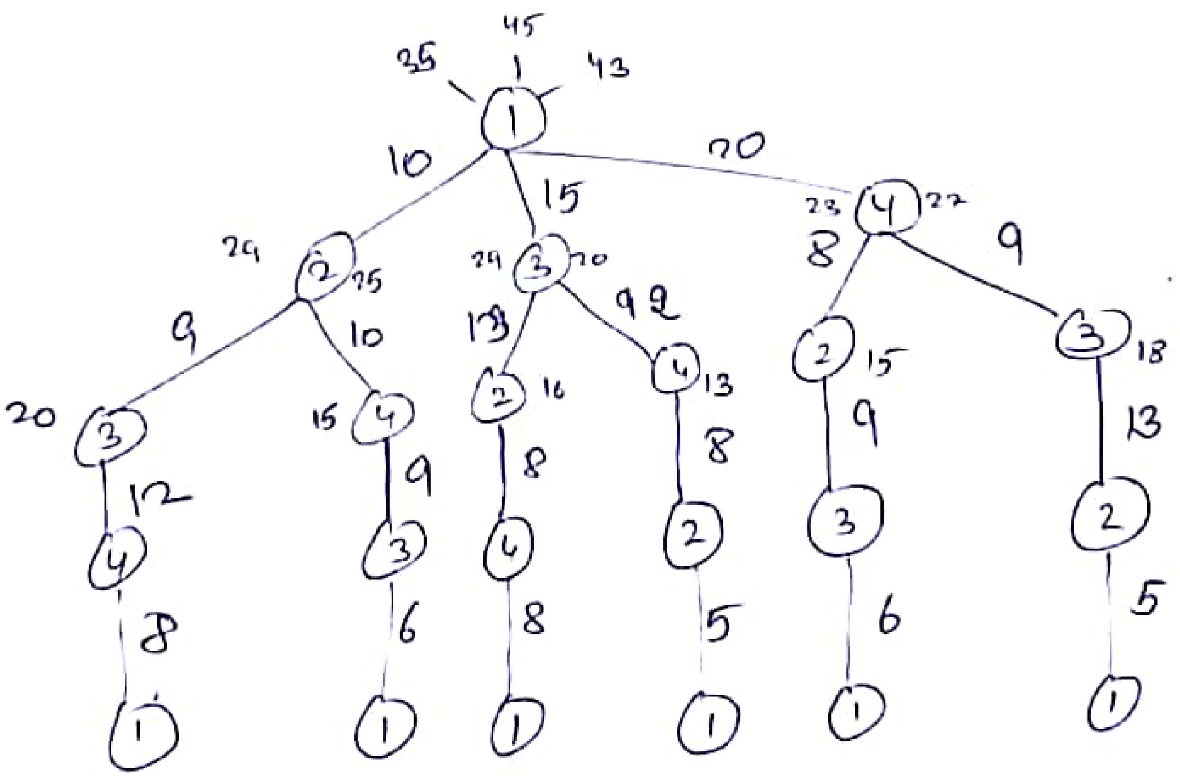
$1 \xrightarrow{4} 6 \xrightarrow{3} 7 \xrightarrow{2} 8 \xrightarrow{4} 5 \xrightarrow{3} 2 \xrightarrow{2} 3 \xrightarrow{1} 4 \xrightarrow{6} 1$

Total = 25

Scanned by CamScanner



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



Min $\{35, 45, 43\} = 35$

$\{1, 2, 4, 3\}$

$$g(i, S) = \text{Min}\{C_{ij} + g(j, S - \{j\})\}$$

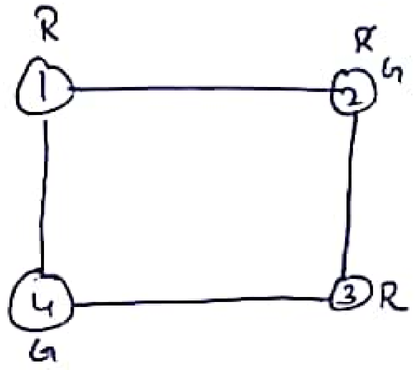
$$S = \{1, 2, 3, 4\}$$

$$\underbrace{10}_{C_{ij}} + 25 = 35$$

Graph Coloring

In graph coloring we have to color the vertices of a graph such that no 2 adjacent vertices must have same color.

Eg - We have given a simple graph and a set of 3 colors let say Red, Blue, Green.

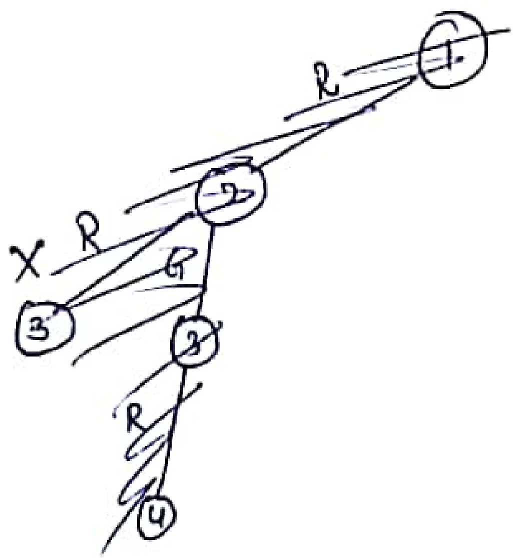


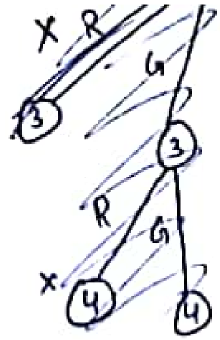
Colors - R, B, G

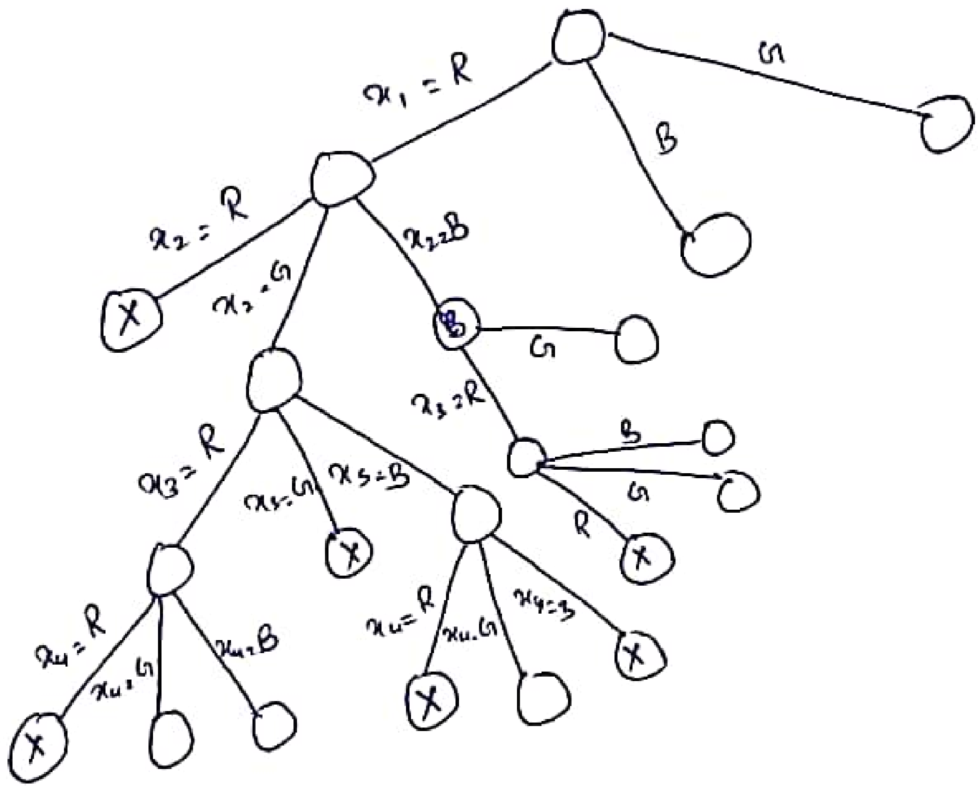
1	2	3	4
R	G	R	G
R	G	R	B

~~We~~ ~~Can~~ colo

There are so many solution possible for this graph. We can find all the solutions using backtracking







Solutions :

RGRG
RGRB
RGBG
RBRG
RBRB

And so on many more solutions can be obtained.

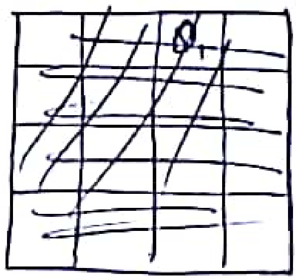
N - Queen Problem

N - Queens problem is to place n - queens in such a manner on an n x n chess board that no two queen attack each other by being in the same row, column, or diagonal.

Eg - We have 4 queens to be placed in 4 x 4 chessboard.

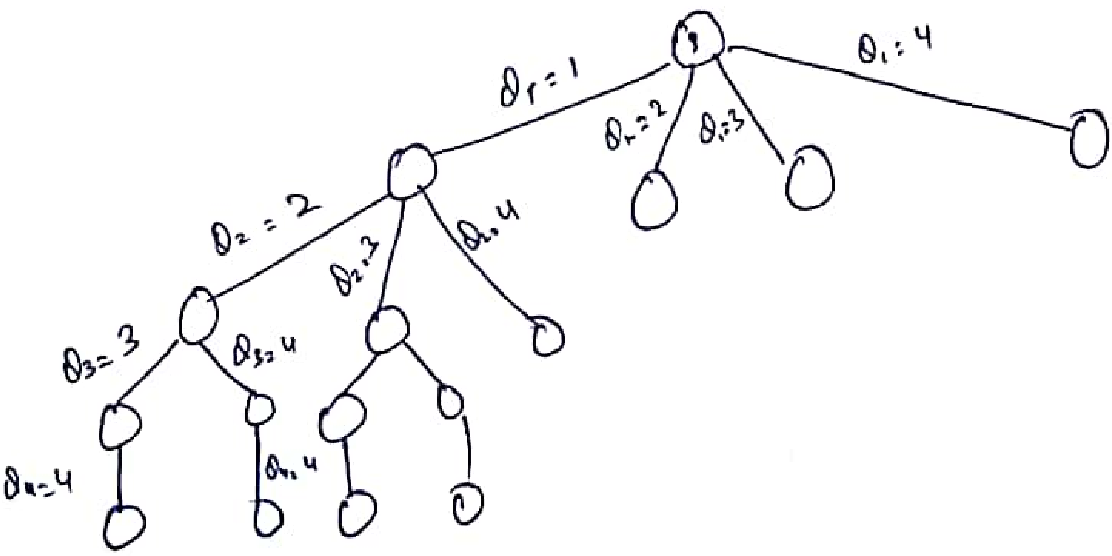
	1	2	3	4
1	Q_1	x		
2	x	x	Q_2	
3	x	x	x	x
4				

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



1	2	3	4
Q_3	Q_1	Q_4	Q_2

There can be more than 1 solution possible. We can use backtracking method to generate next node and stop if next node violet the rule.



Number of nodes possible

$= 1 + 4 + 4 \times 3 + 4 \times 3 \times 2 + 4 \times 3 \times 2 \times 1$

$= 65$

$$= 1 + \sum_{i=1}^{n} \left[\prod_{j=0}^{i} (4-j) \right]$$

Algorithm –

```
place (k, i)
{
    for (j=1 to k-1)
    {
        if ((x[j] = = i) or (Abs (x[j]-i) =
                                Abs (j-k)))
            then
                return false
    }
    return true;
}

NQueens (k, n)
{
    for (i=1 to n)
    {
        if (place (k,i))
        {
            x[k] = i;
            if (k = = n)
```

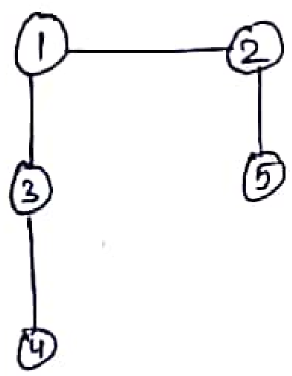
```
then
    print (x[1:n])

else
    NQueens (k+1, n)
  }
 }
}
```

Hamiltonion Cycle

Hamiltonion Path -

Given a graph $G = (V, E)$ we have to find the Hamiltonion path.

Eg -



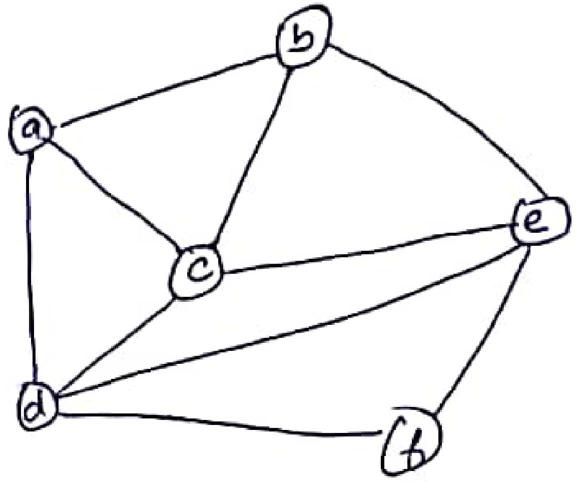
Path - 4, 3, 1, 2, 5
or
5, 2, 1, 3, 4.

Visiting each vertex ~~atleast~~ only once ~~with~~ is called hamiltonion path.

Hamiltonion cycle - visiting each vertex only once with same starting and ending vertex is called hamiltonion cycle.

Hamiltonion using backtracking

Eg -

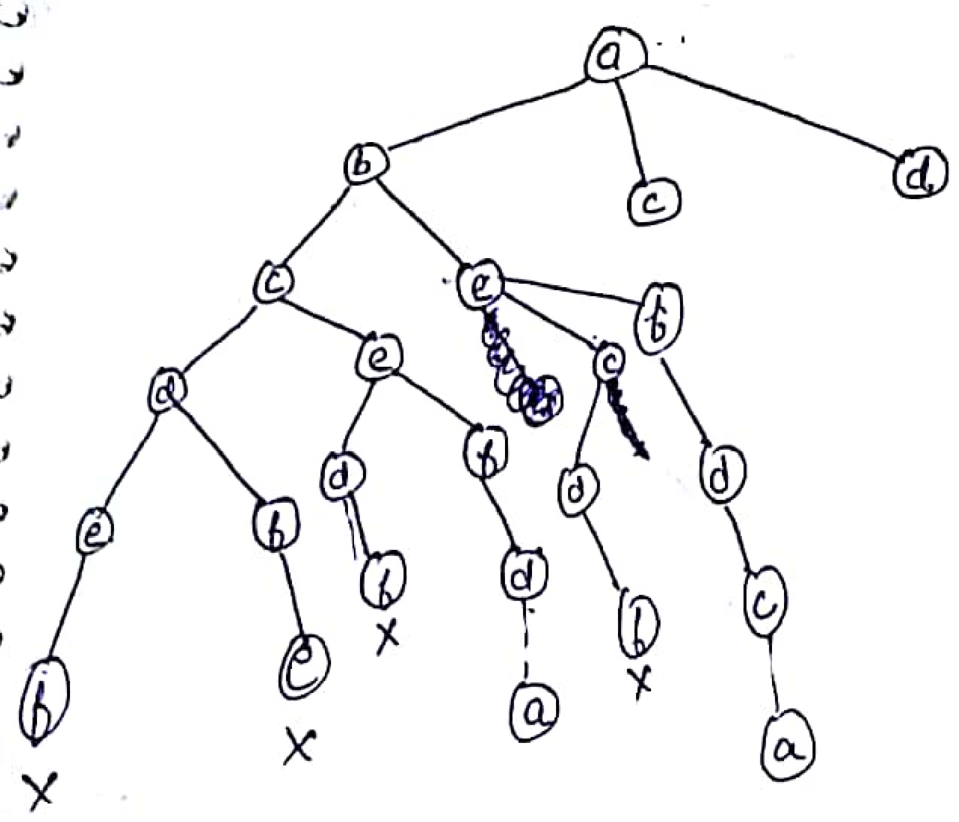


We have to create an adjacent matrix having ~~ra~~ rows and column equal to number of nodes.

	a	b	c	d	e	f
a	0	1	1	1	0	0
b	1	0	1	0	1	0
c	1	1	0	1	1	0
d	1	0	1	0	1	1
e	0	1	1	1	0	1
b	0	0	0	1	1	0

Conditions for construction of cycle

- no repeation of value
- if we are inserting $x[k]$ it must be connected to $x[k-1]$
- last node and first node must have a edge in between.



1	2	3	4	5	6
a	b	c	e	f	d.

1	2	3	4	5	6
a	b	e	f	d	c

and so on more solutions can be obtained.

Algorithm -

```
NextVertex(k)
{
    do
    {
        x[k] = (x[k] + 1) Mod(n + 1)
        if (x[k] = 0)
            return;
```

```
        if (G (x[k-1] , x[k]) ≠ 0)
        {
            for (j: 1 to k -1)
            {
                if (x[j] = x[k])
                    break ;
                if (j = k)
                    if (k < n or (k = n &&
                                  G (x[n] , x[1]) ≠ 0))
                        return ;
            }
        }
    }
    while (true) ;
}
Hamiltonian (k)
{
    do
    {
        NextVertex (k);
        if (x[k] = 0)
            return ;
        else if (k == n)
            print (x[1:n]);
        else
            Hamiltonian (k+1);
    }
    while (true);
}
```