Unit - 3

Greedy Algorithm & Dynamic Programming

Greedy Algorithm :-

A greedy Algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.

It basically based on "take whatever you can get" strategy.

Greedy is a strategy that works well on optimization problems with following charackristics :-

(a) Greedy-choice property :- A global optimum can be arrived at by selecting a local optimum

(b) optimal substructure :- An optimal solution to the problem contains an optimal solution to subproblem.

Examples of Greedy Algorithm :-

→ Huffman Coding
→ Fractional Knapsack Problem
→ Kruskal's algorithm for Minimum Spanning Tree.
→ Prim's algorithm for Minimum Spanning Tree (MST).

Huffman Coding :-

Huffman Coding is a lossless data compression algorithm. The idea is to assign variable-length codewords to input characters, lenghths of of codes are based on frequency of corresponding characters.

The most frequent character gets the smallest code and the least frequent character gets the largest code.

Note → The codes are assigned in such a way that the code assigned to one character is not a prefix of ~~any~~ code assigned to any other character. This is how huffman coding make sure that there is no ambiguity when decoding the generated bit stream.

Q.

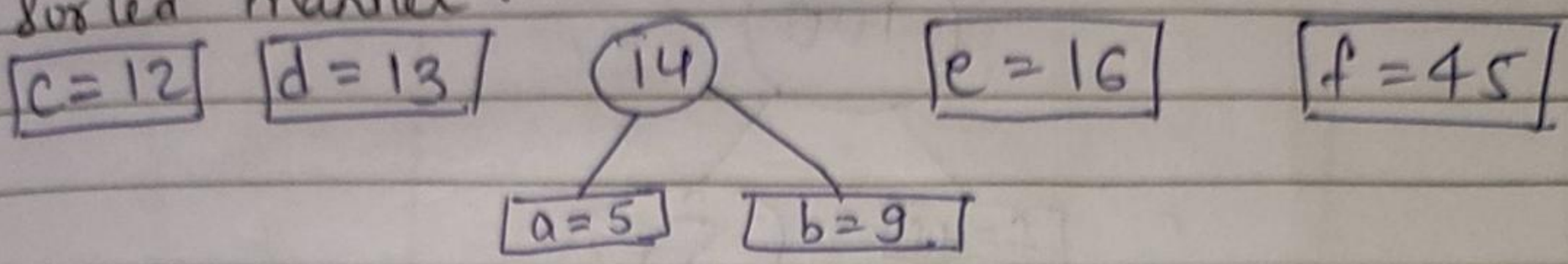
Character	Frequency
a	5
b	9
c	12
d	13
e	16
f	45

By using Huffman Coding find codeword for each character.

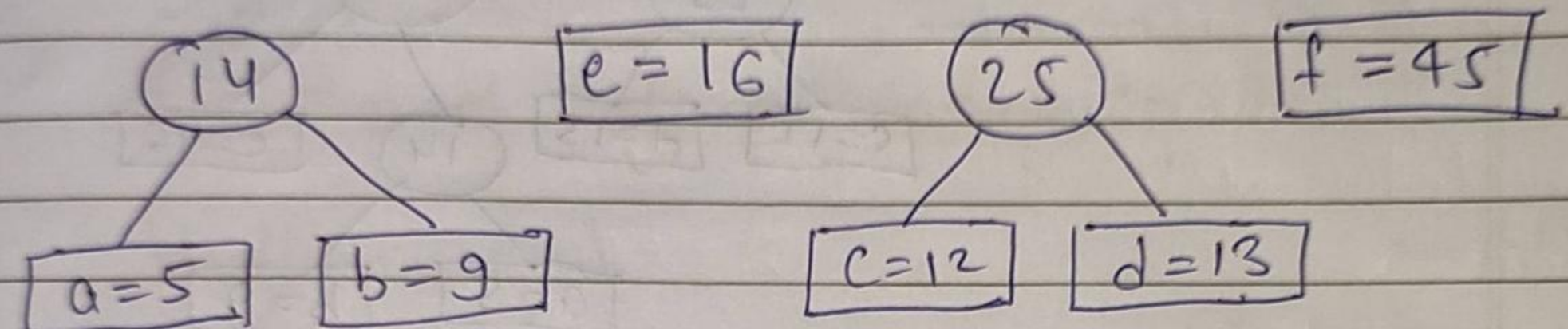
Ans → Step 1 :- Arrange all the character's in ascending order :-

a	b	c	d	e	f
5	9	12	13	16	45

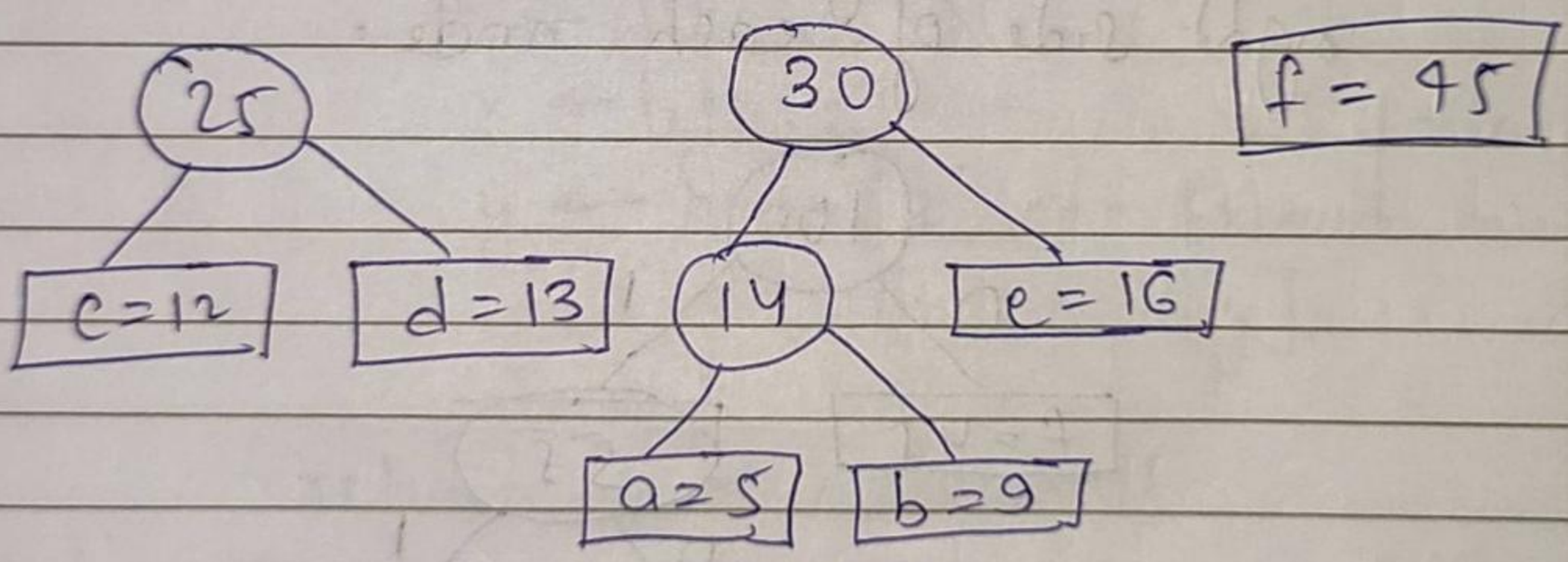
Step 2 :- Merge two smaller order frequency i.e a and b in this case and put it in sorted manner.



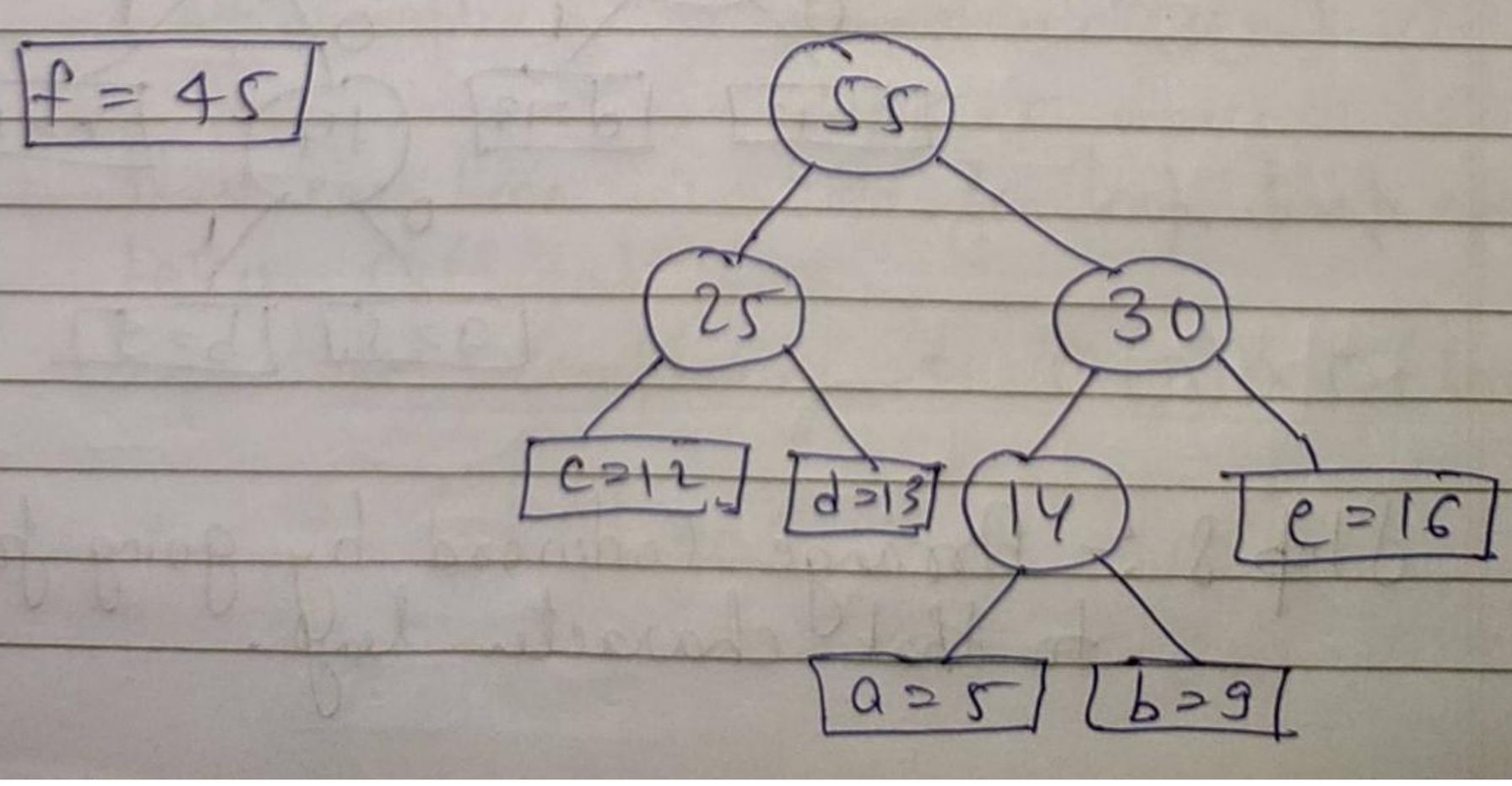
Step 3 :- Repeat step 2 :- c and d in this case.



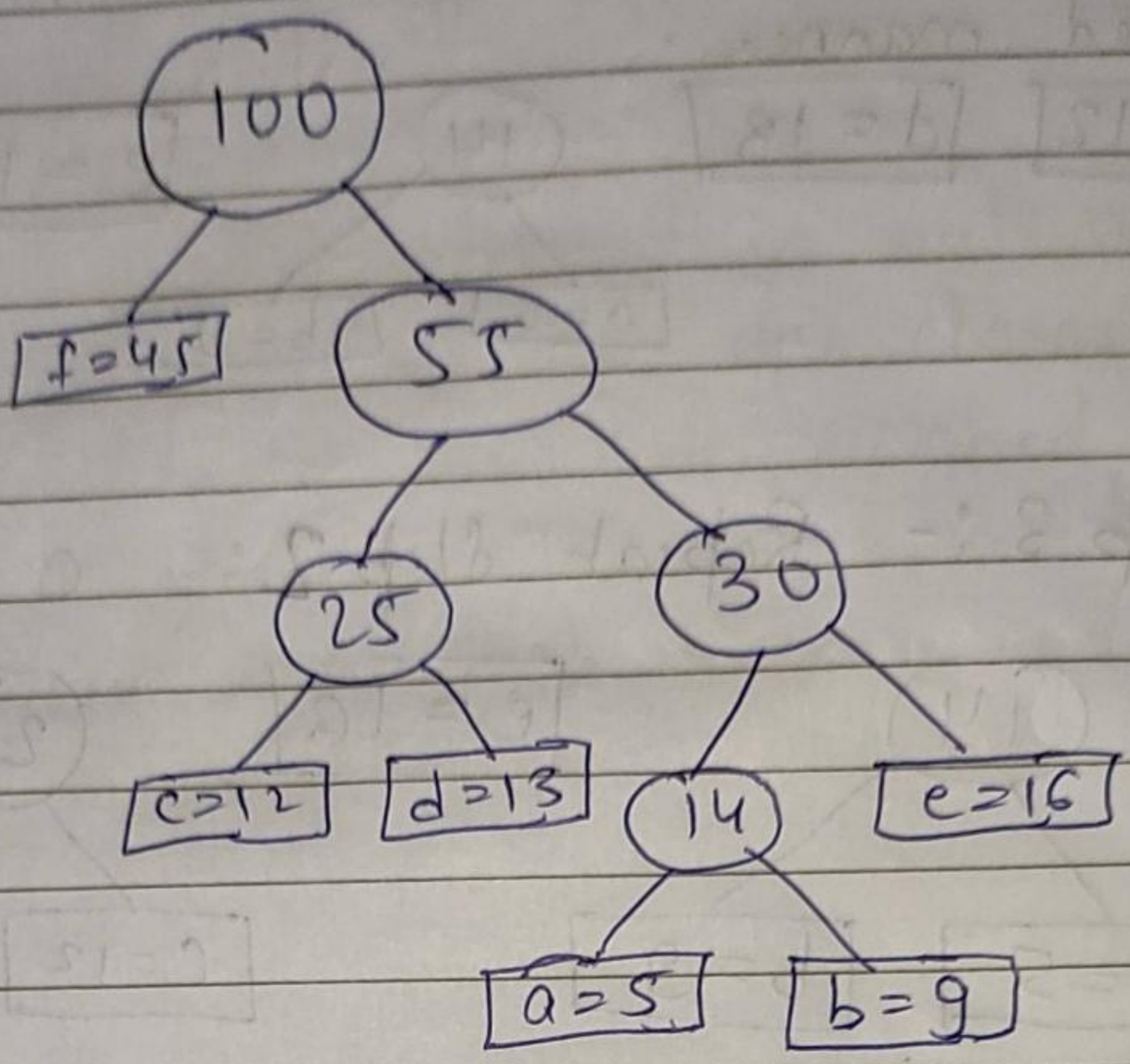
Step 4 :- Repeat step 2 :- 14 & 16 this time.



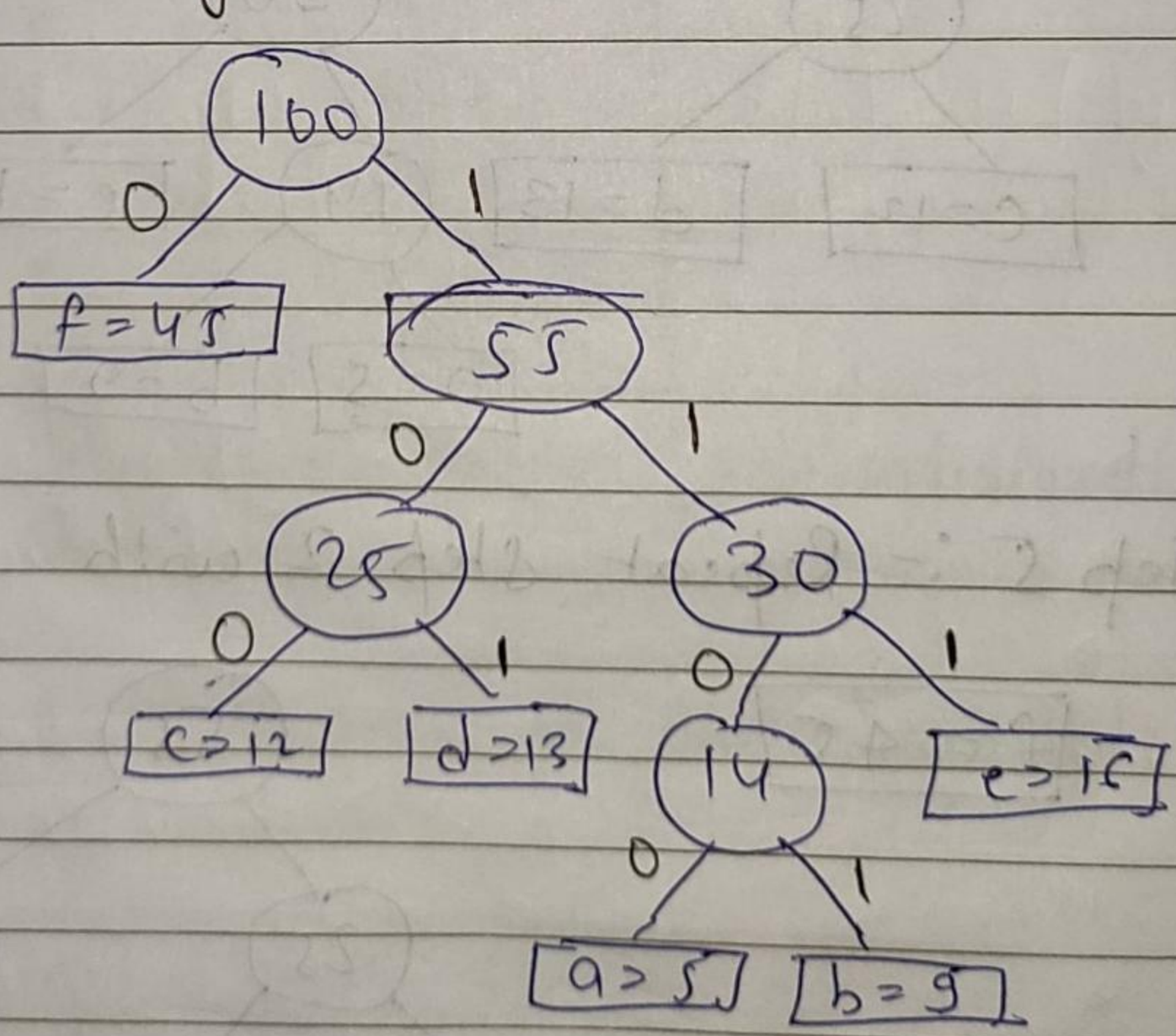
Step 5 :- Repeat step 2 with 25 & 30 this time



Step 6 :- Repeat step 2 with 45 & 55 this time



Step 7 :- Now, when tree is complete assign 0 to the left side and 1 to the right side of each node.



Step 8 :- Arrange codeword by going from root to that character leaf.

char	frequency	Codeword
a	5	1100
b	9	1101
c	12	100
d	13	101
e	16	111
f	45	0

Ans

Algorithm of Huffman coding :-

```
Huffman (c)
    n ← |c|
    Q ← c
    for i ← 1 to n-1 do
        z ← Allocate_Node ()
        x ← left[z] ← Extract_min(Q)
        y ← Right[z] ← Extract_min(Q).
        f[z] ← f[x] + f[y].
        Insert (Q, z).
    return Extract_min(Q).
```

Analysis :-

* for loop will run n-1 times. So for for loop complexity is of order O(n).

Now, each time inside the for loop, heap operation takes O(logn) time.
So, total complexity = O(n) x O(logn).

= O(nlogn)

Q.

char	frequency
a	4
b	5
c	7
d	8
e	10
f	12
g	20

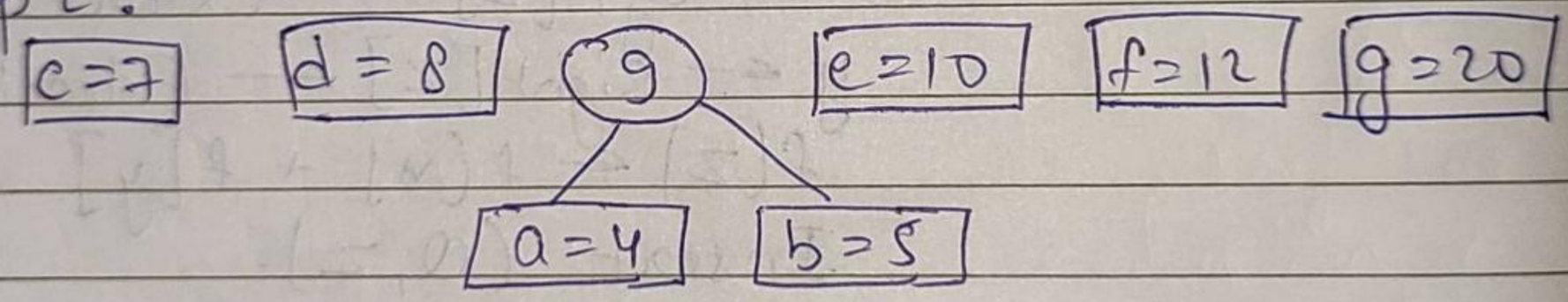
find codeword of each character using Huffman coding.

Ans →

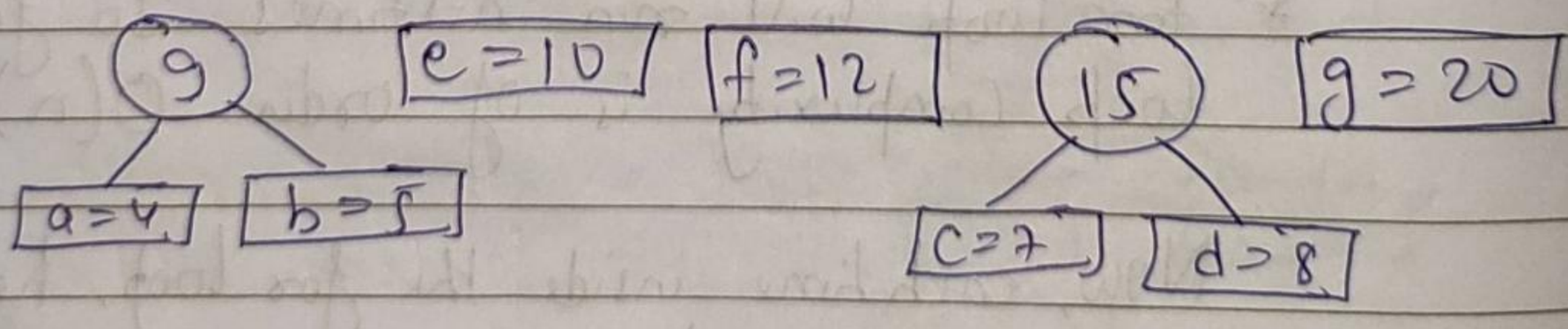
Step 1 :-

a	b	c	d	e	f	g
4	5	7	8	10	12	20

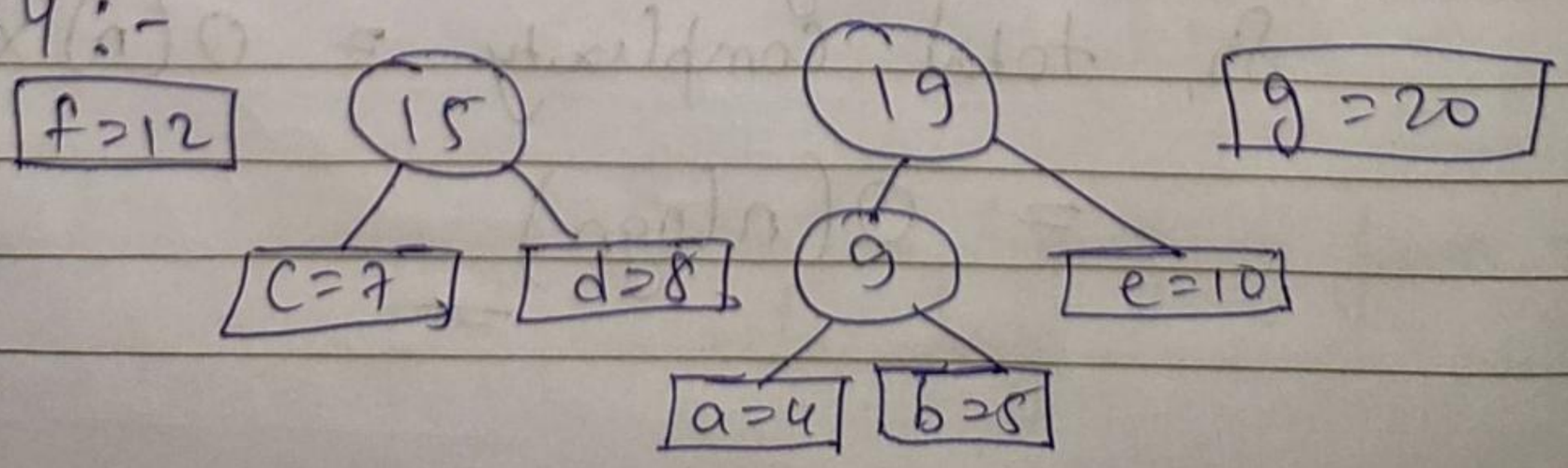
Step 2 :-



Step 3 :-

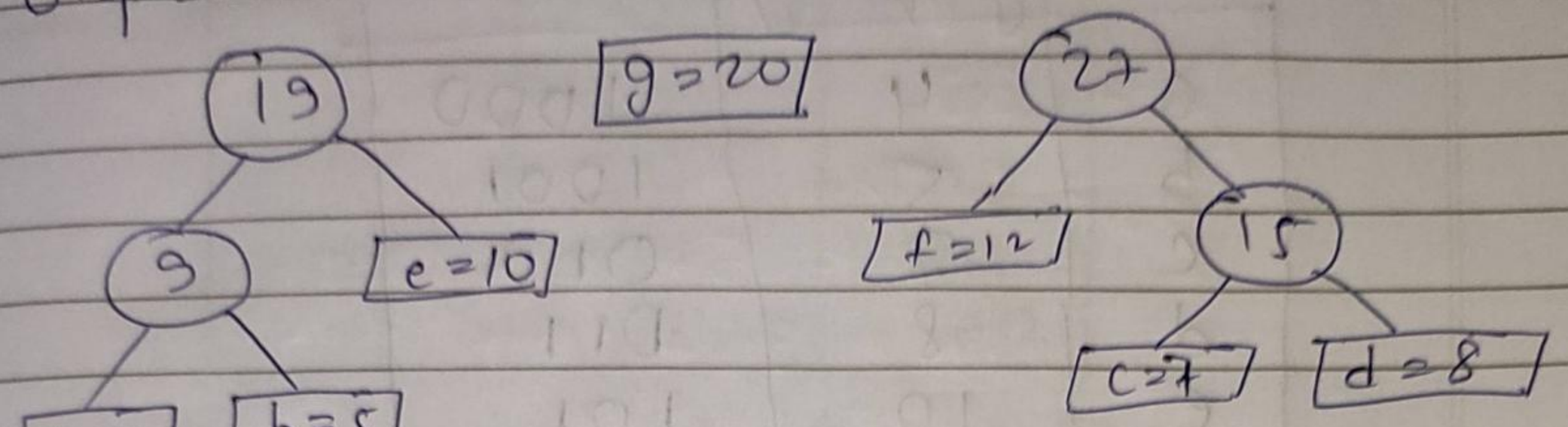


Step 4 :-

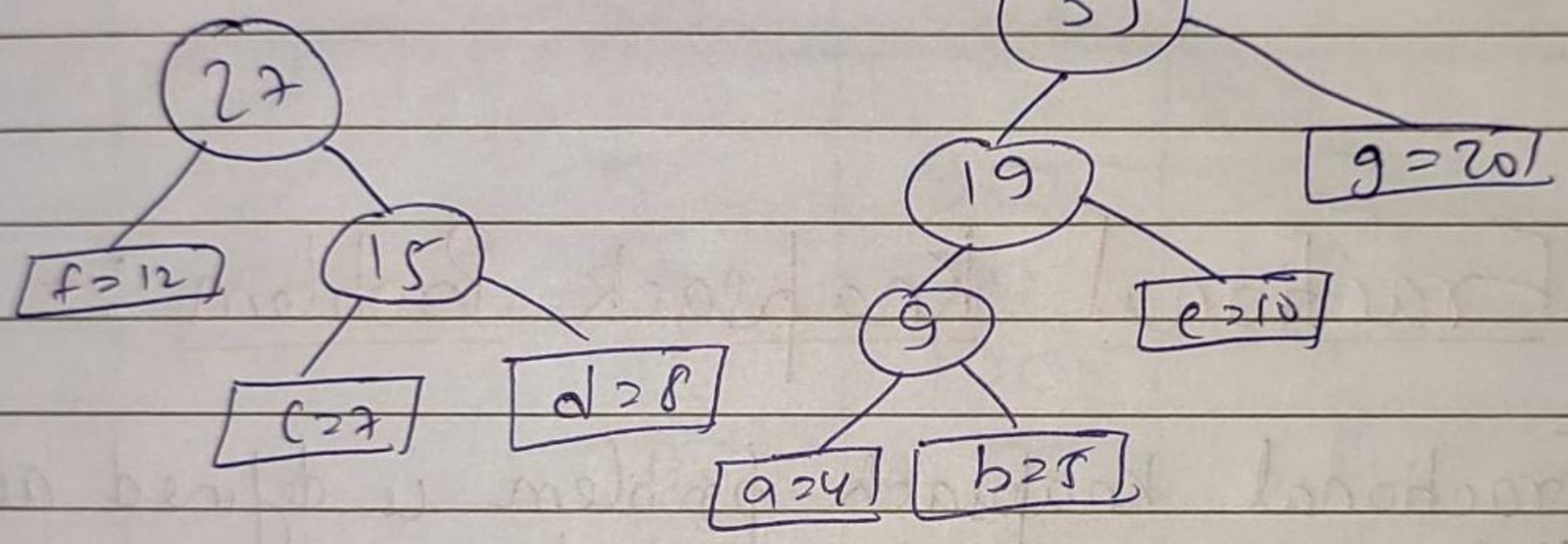


Date
Page

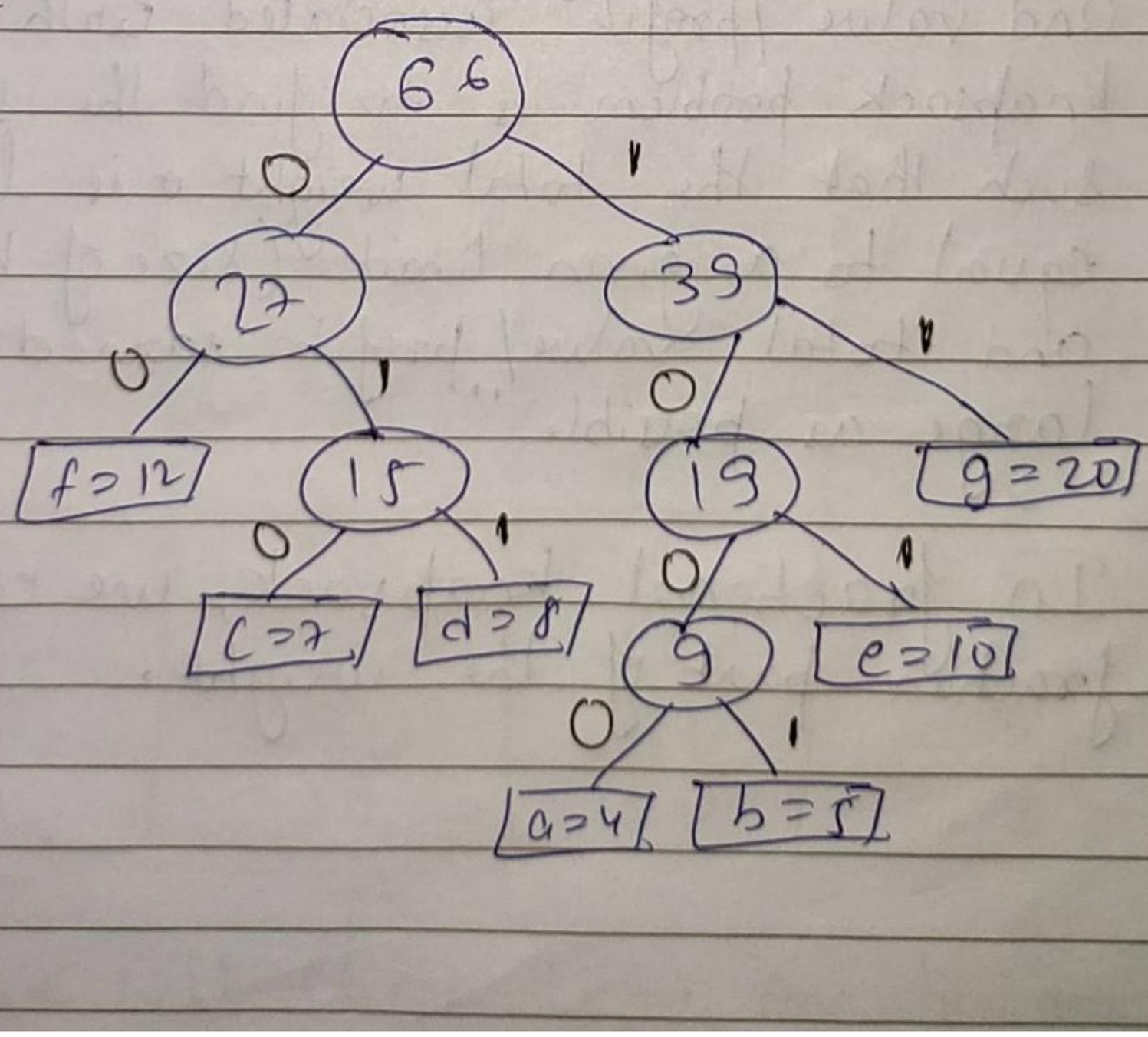
Step 5 :-



Step 6 :-



Step 7 :-



Char	frqiency	Codeword
a	4	1000
b	5	1001
c	7	010
d	8	011
e	10	101
f	12	00
g	20	11

Ans

Fractional Knapsack Problem :-

Fractional knapsack problem is defined as "Given a set of items having some weigth and value /profit associated with it. The knapsack problem is to find the set of items such that the total weight w is less than or equal to a given limit (size of knapsack) and total value/profit earned is as large as possible".

In fractional knapsack, we can take fraction part of the weight.

Q.

Items	Weight	Value
I_1	5	30
I_2	10	20
I_3	20	100
I_4	30	90
I_5	40	160

Given knapsack weight = 60 kg.

Ans →

Items (I_i)	Weight (w_i)	Value (v_i)	$P_i = \frac{v_i}{w_i}$
I_1	5	30	6
I_2	10	20	2
I_3	20	100	5
I_4	30	90	3
I_5	40	160	4

Now, we sort it according to P_i value in decreasing order.

Items	Weight	Value	$P_i = \frac{v_i}{w_i}$
I_1	5	30	6
I_3	20	100	5
I_5	40	160	4
I_4	30	90	3
I_2	10	20	2

Now, we will chose Items one by one while checking if weight is greater than 60.

$I_1 = 5kg < 60kg$

Value = 30.

$I_2 = 20kg$

weight $= 5+25 \leq 60kg$

$\Rightarrow$ value $= 30 + 100$

$= 130$

$I_3 = 40$

weight $= 40+25 \geq 60$

$\downarrow$

Since it is greater, we will take fraction part of it (35 in this case)

$I_3 = 35$

Total weight $= 35+25 = 60kg$

Value $= 130 + 35 \times \frac{160}{40}$.

$= 130 + 140$

$= 270$

So Maximum possible value = 270
& I_1, I_2 & fraction of I_3 are involved.

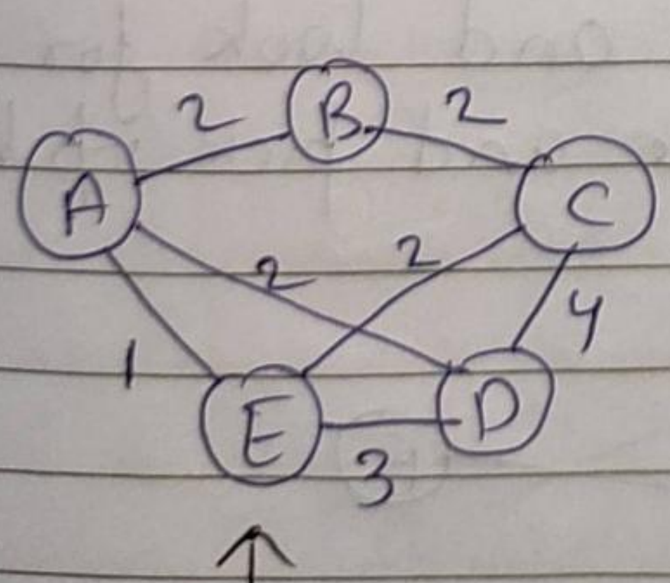
Ans

Algorithm of fractional knapsack :-

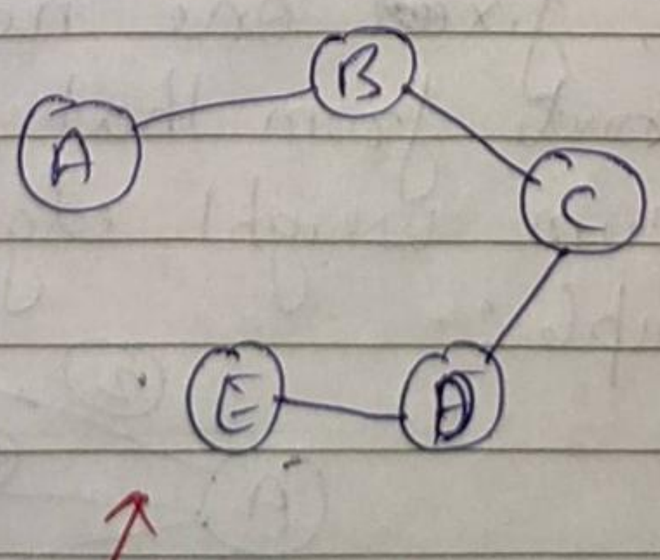
```
fractional knapsack
    for i = 1 to size(v)
    do p[i] = v[i] / w[i].
    Sort-descending (p).
    i = 1
while (w > 0)
do
    amount = min (w, w[i])
    solution[i] = amount
    w = w - amount
    i ← i + 1
return solution
```

Prim's Algorithm for MST :-

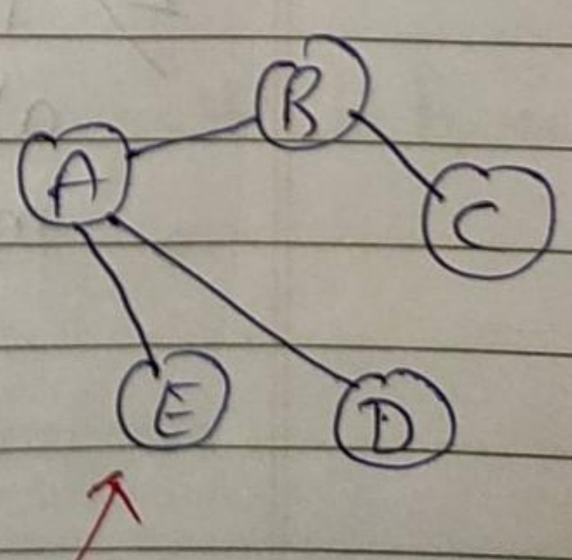
* Spanning Tree :- Connected graph without any cycle.
* Minimum Spanning Tree :-



Connected graph with cycle



Connected graph without cycle



Minimum Spanning Tree

Now, to solve MST, we have two methods :-

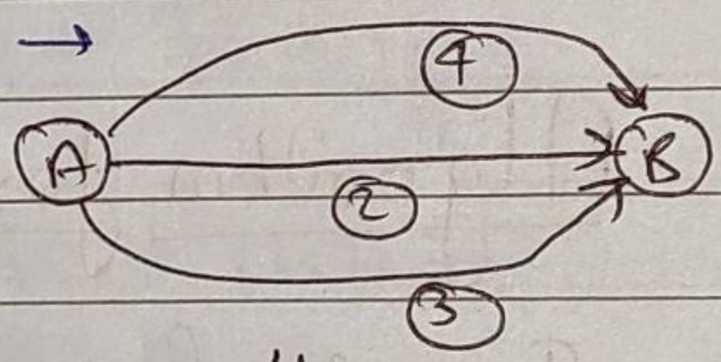
i) Prim's Algorithm
ii) Kruskal's Algorithm

In this, we will talk about Prim's Algorithm :-

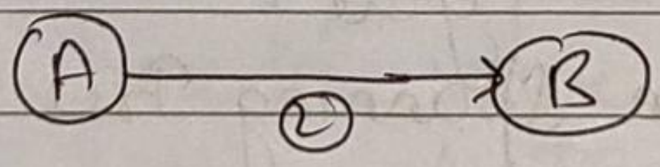
Procedure to solve Problem :-

1) If there are more than 1 edge in a graph, then remove all except the minimum weigh graph.

example →

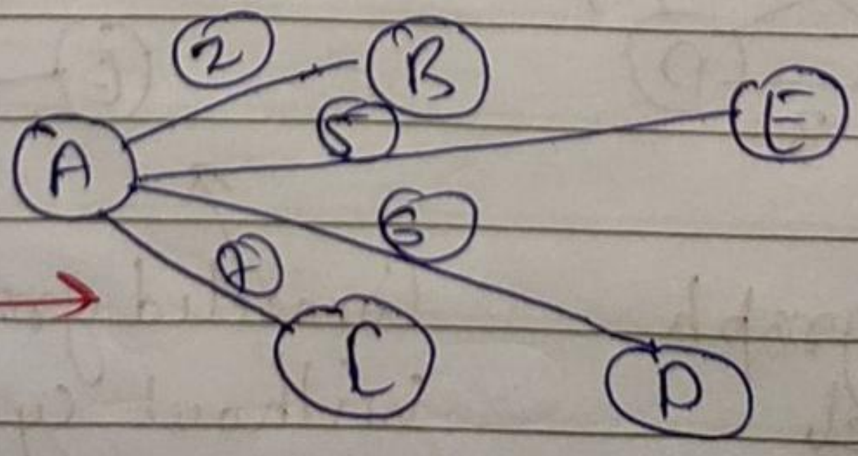


Remove edge of weight 3 & 4 →



2) Now, fixed one node, and look for all route from that node and pick up the minimum weight edge.

example :-

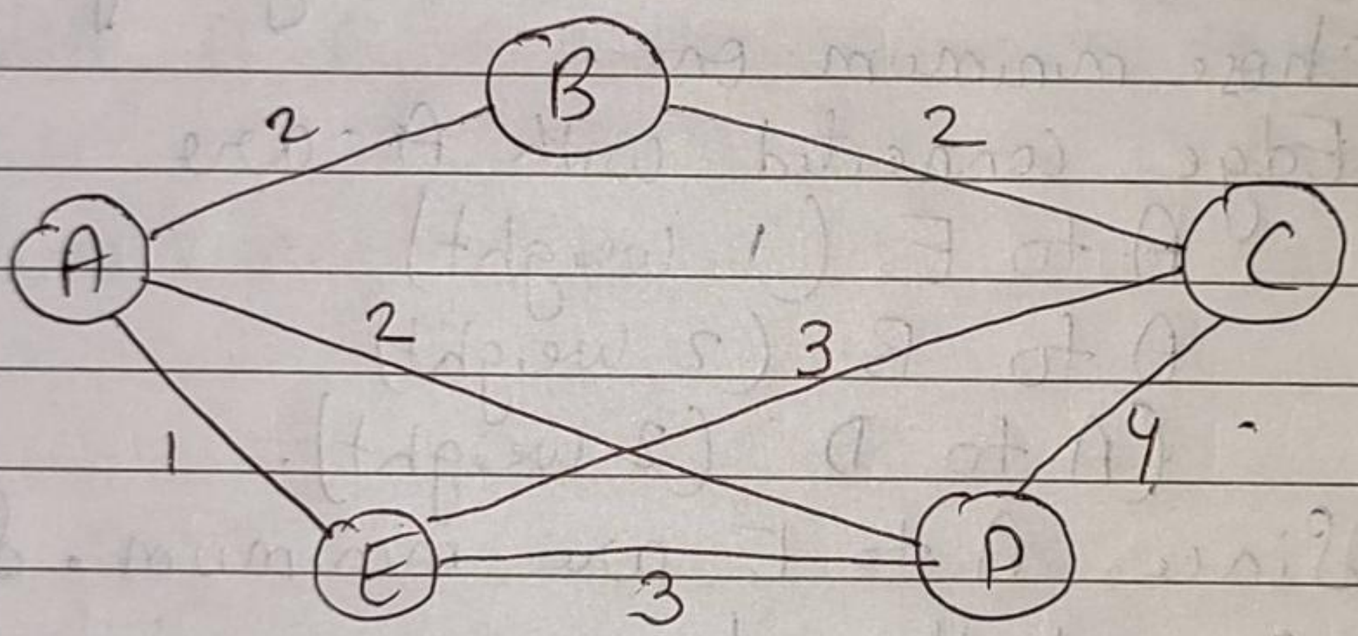


We will pick A to B →

3) Repeat all the skp 2 with all the connecting node.

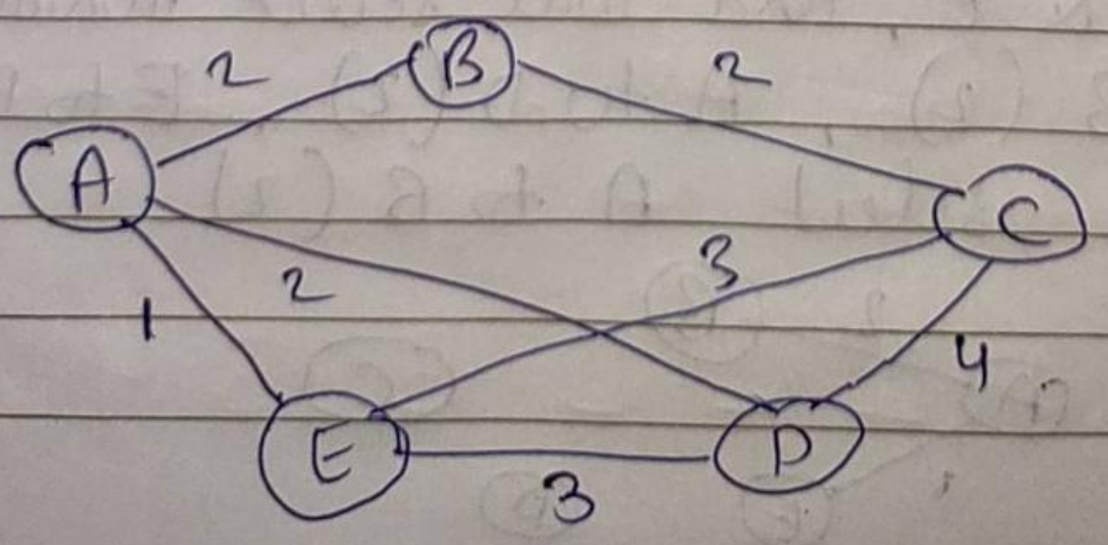
(जिसका जिसका Node connect हो रहा है, उसमें से check करना है कि कौन सा Edge में Minimum weight है, and उसको जोड़ देना है)

Q. Find Minimum Spanning Tree of this given graph using Prim's algorithm.

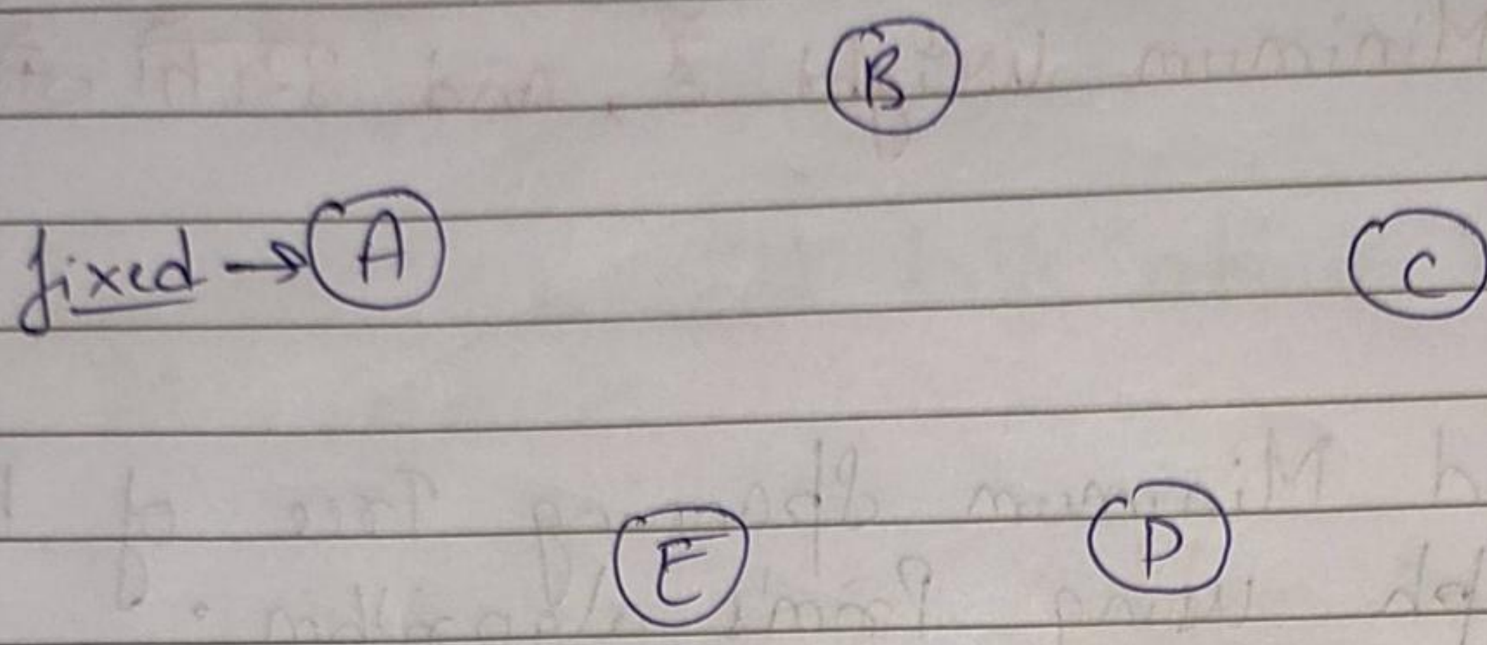


Ans → * [Note → I will explain how, we will write this ques in exam. After this Question, I will not show entire process again and again]

Step 1 :- Remove all alternative edge path between two nodes (if available).



Step 2 :- Since, there is no fixed node given, we can take any node of our choice and fix it.



Step 3 :- Now, we see all the edge of A and chose minimum one.

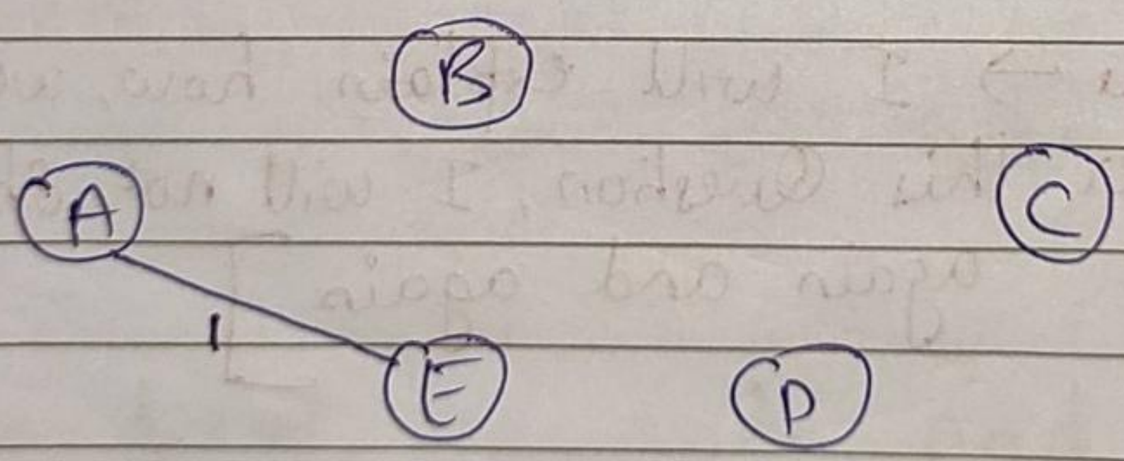
Edge connected with A are

A to E (1 weight)

A to B (2 weight)

A to D (2 weight).

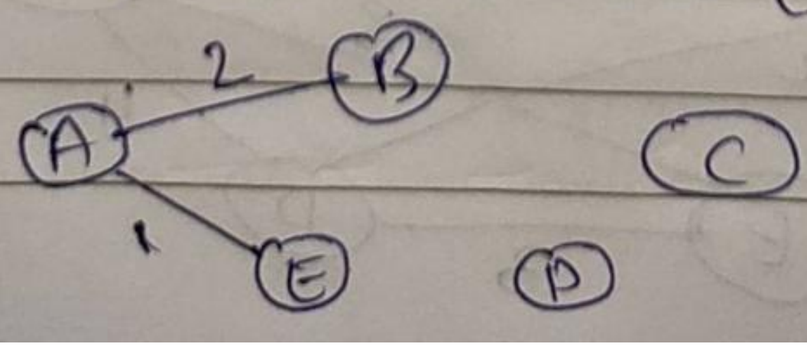
Since A to E are minimum. So, we will connect these two.



Step 4 :- Now, we will see all connected with A & E, and will select minimum one

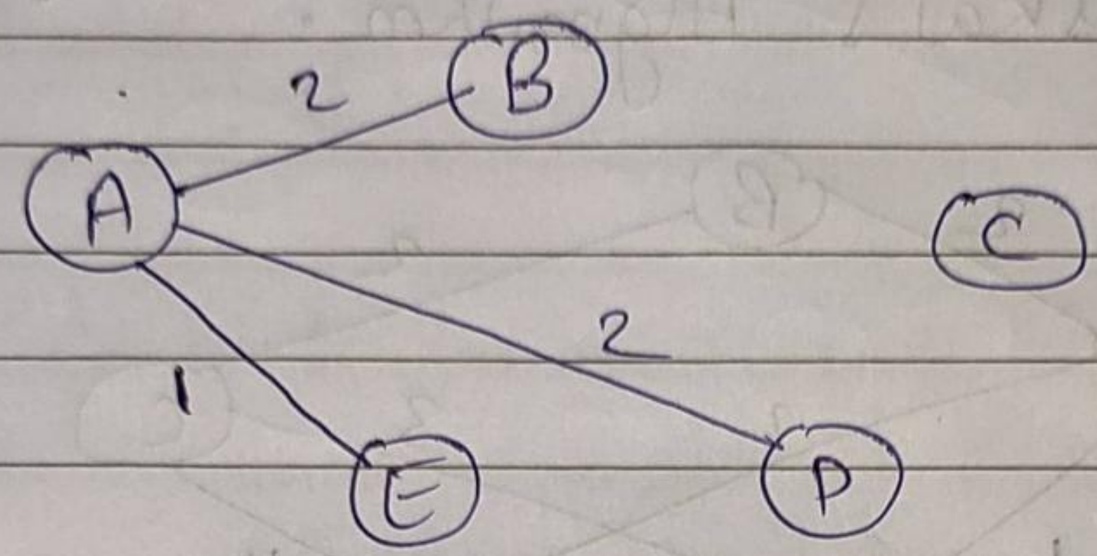
A to B (2), A to D (2), E to D (3), E to C (3)

we select A to B (2).



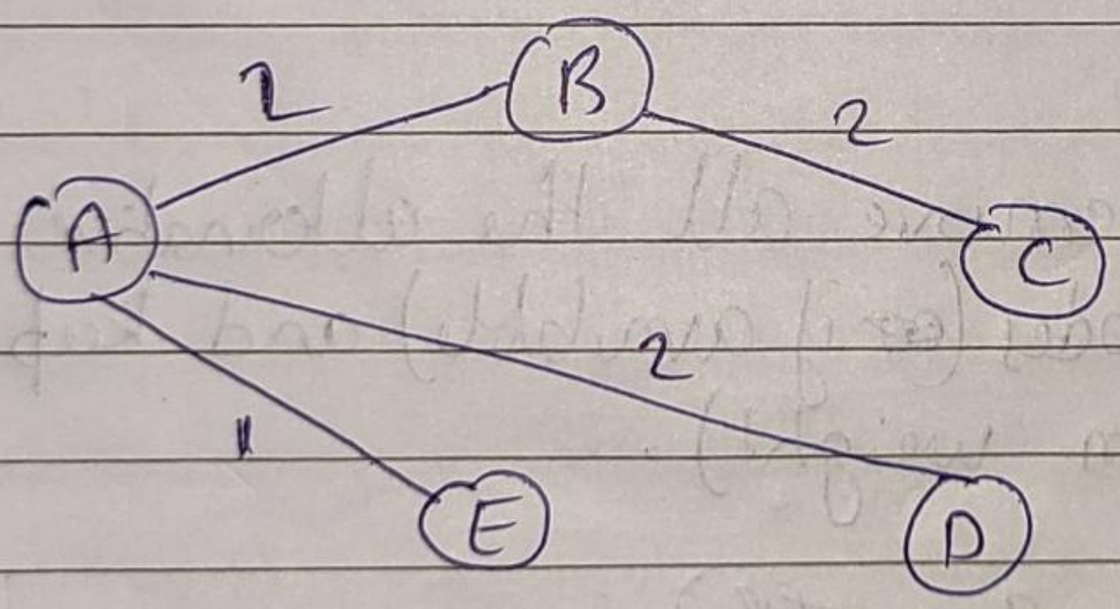
Step 5 :- Now, we will choose from all edge conceted with A, B and E. i.e [A to D(2), B to C(2), E to D(3), E to C(3)]

we select A to D(2).



Step 6 :- Now, we will choose from all edge conceted with A, B, E and D. i.e [B to C(2), E to D(3), E to C(3), C to D(4)]

we select B to C (2)

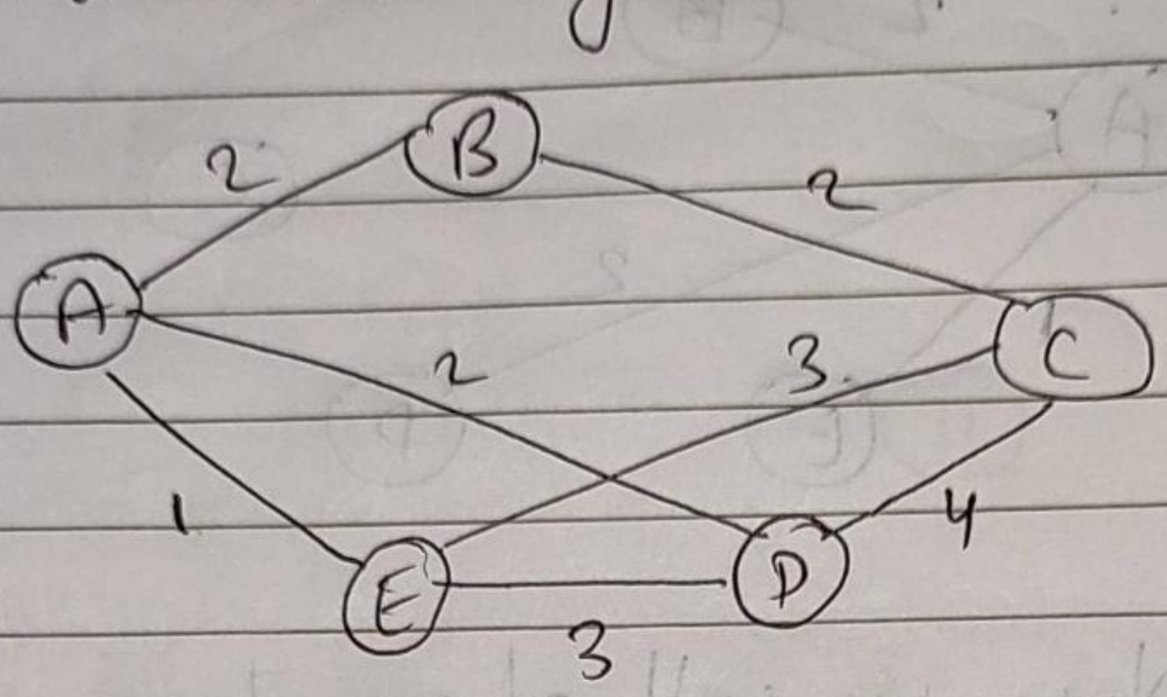


Now, all nodes is conceted, so this is required minimum spanning tree.

Weight of MST = 1+2+2+2
= 7 Ans

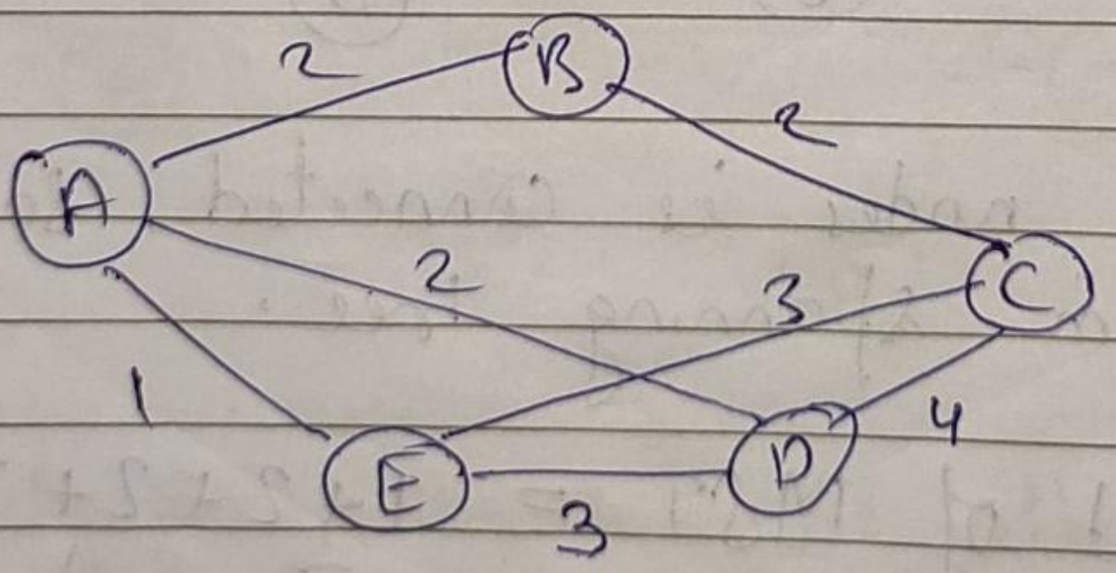
Kruskal's Algorithm for MST :-

Q. Solve following graph to find MST using Kruskal's Algorithm.



Ans → [Note → I will show proper solution which we will write in exam only for this Question]

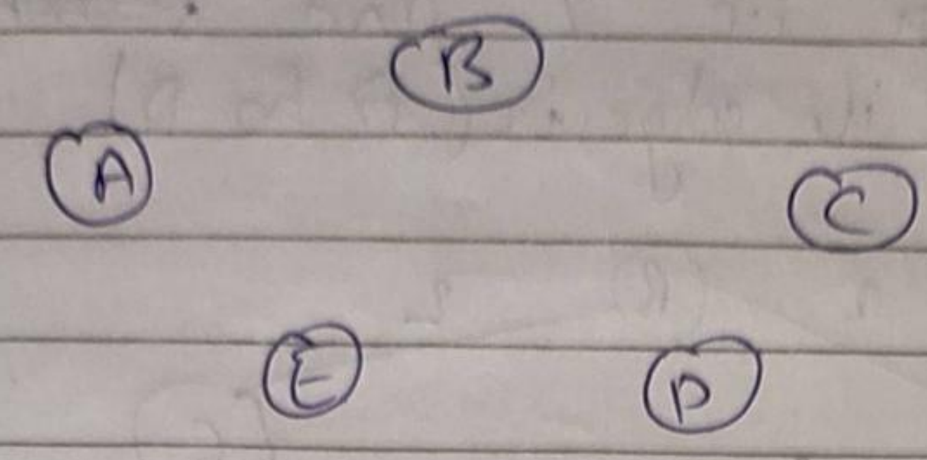
Step 1 :- Remove all the alternative edge between two nodes (or if available) and keep edge with minimum weight).



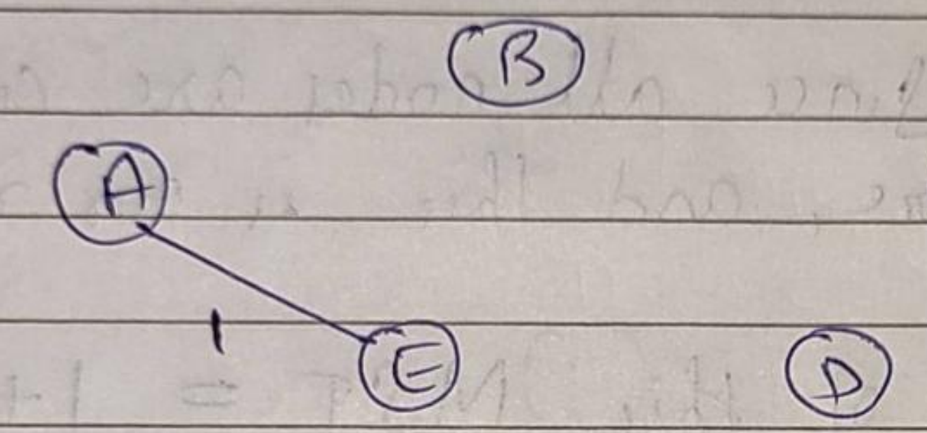
Step 2 :- Arrange all the weight in ascending order

1 2 2 2 3 3 4.

Step 3 :- Writ all node in without any connection

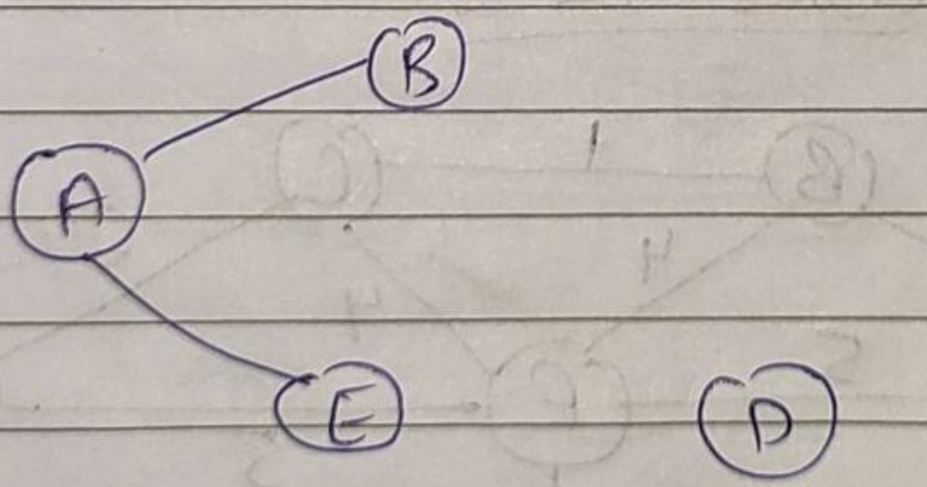


Step 4 :- Select 1 from ascending order and connect node with weight of 1 (i.e A to E)



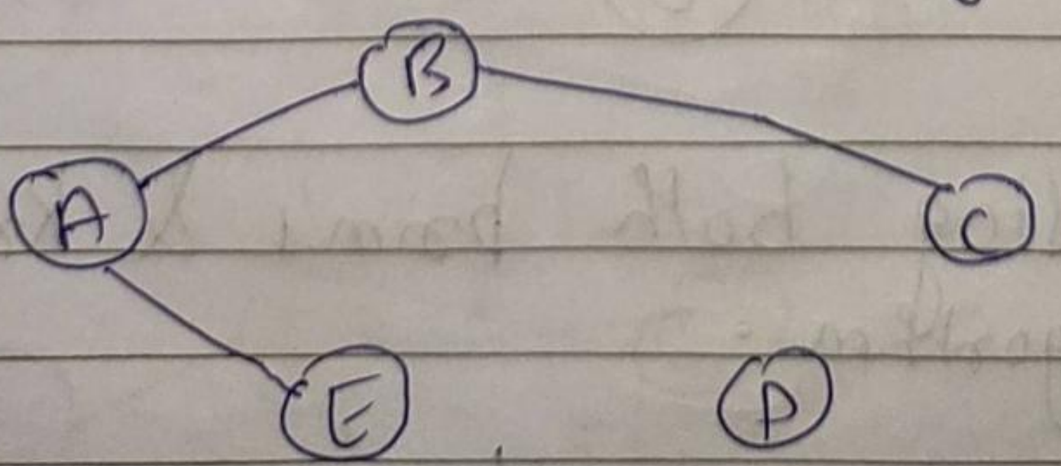
XXX222334

Step 5 :- Now, select next number from ascending order i.e. 2 and connect node with that weight i.e. (A to B)



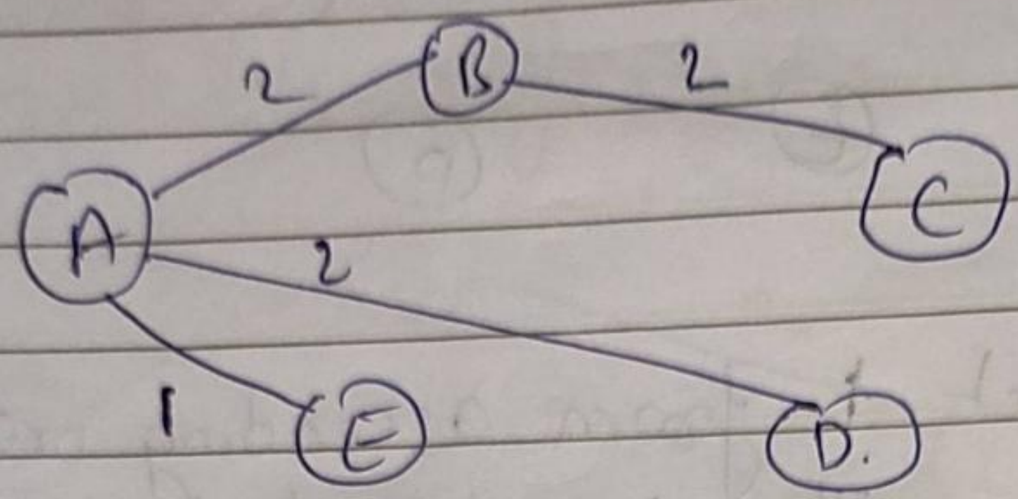
XXX22334

Step 6 :- Now, select next ascending order number i.e 2 and connect edge of this weight (B to C)



XXX2334

Step 7:- Now, select next ascending order number i.e 2 and ~~select~~ connect node with it edge i.e (A to D)



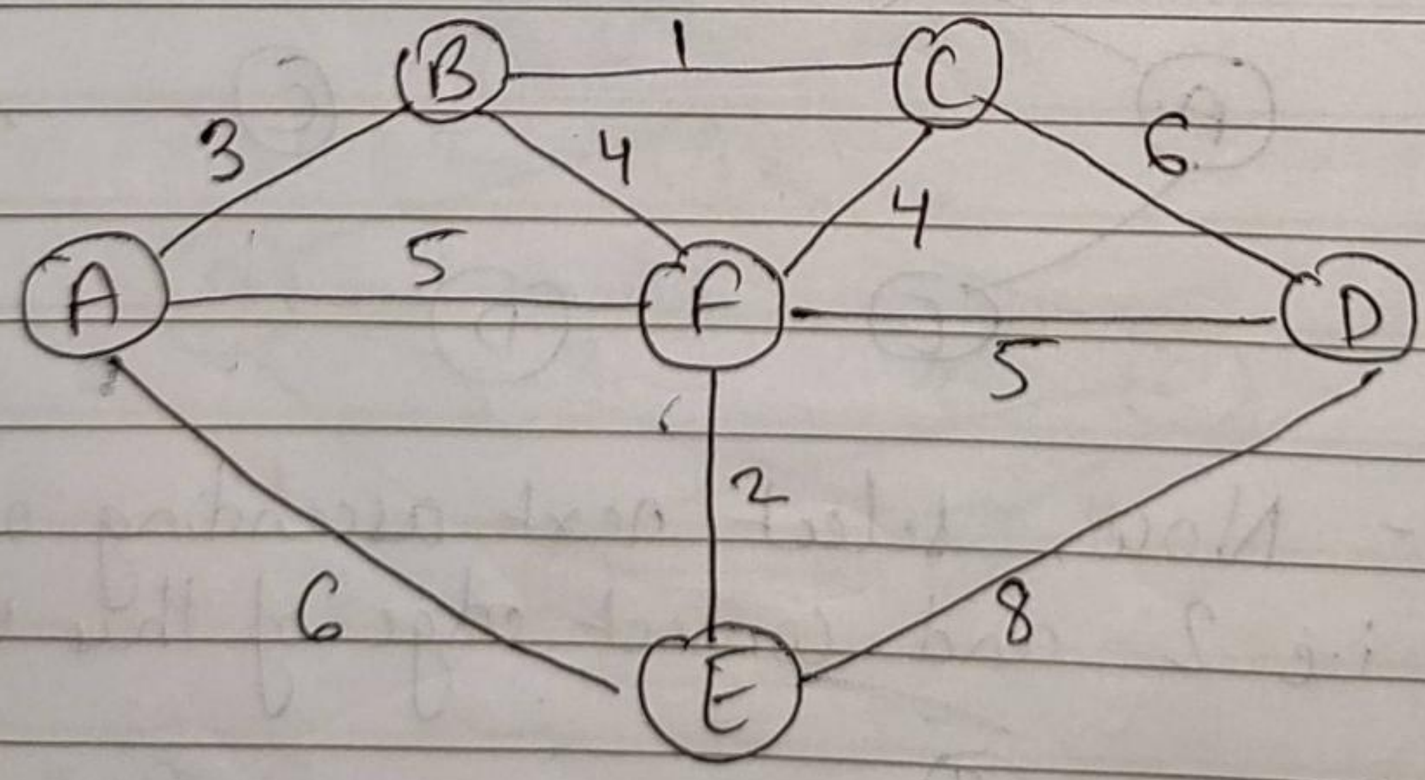
~~Ste~~ Now, since all nodes are connected we stop here, and this is our required MST.

Weight of this MST = 1+2+2+2
= 7.

Ans

Practise Questions :-

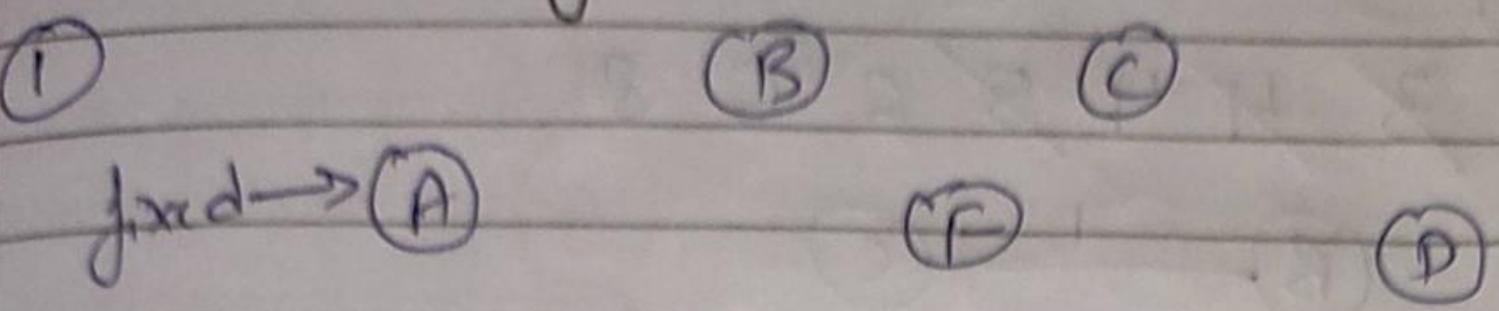
1)



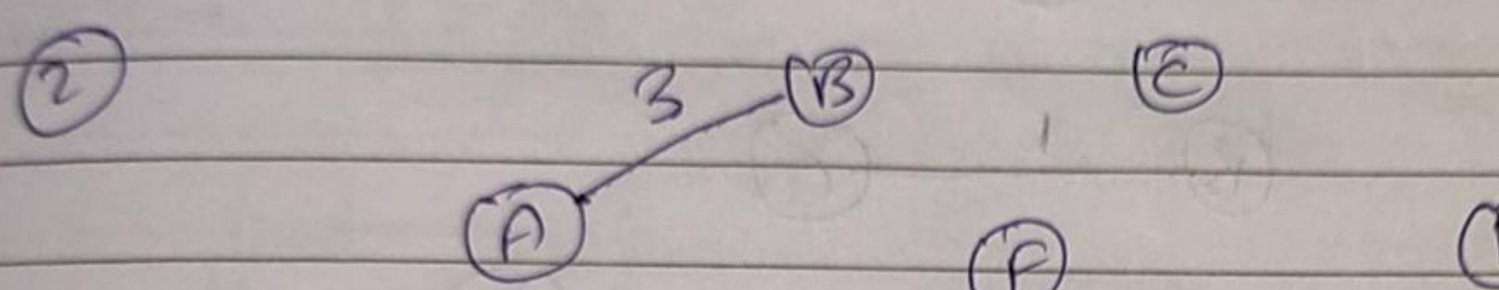
Solve using both Prim's & Kruskal's algorithm.

Sol → Using prim's

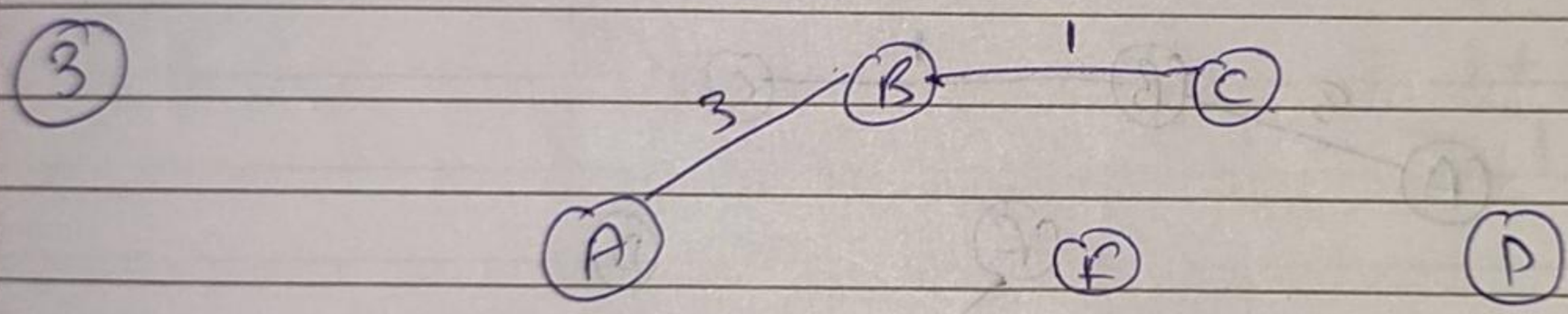
(1)



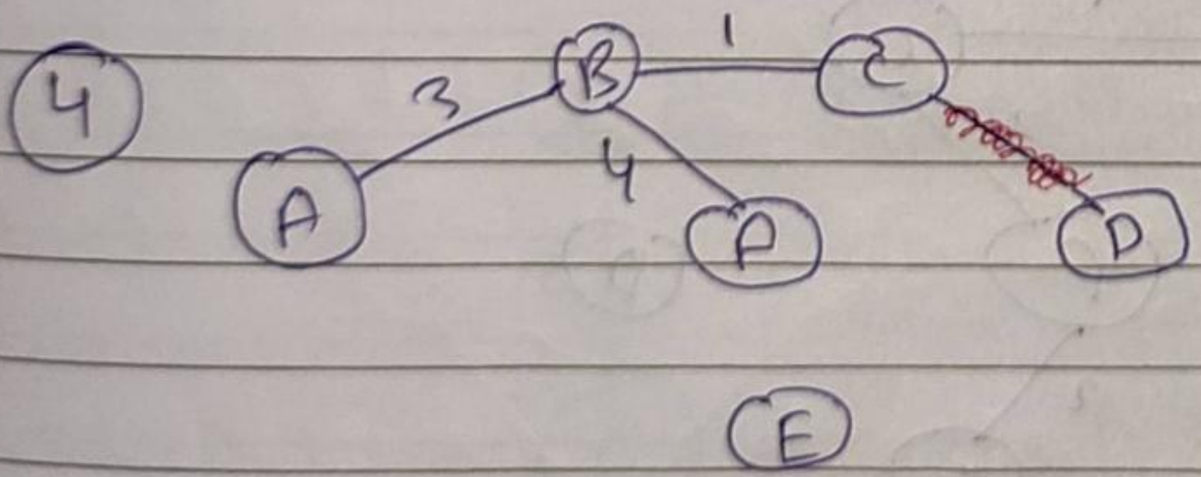
(2)



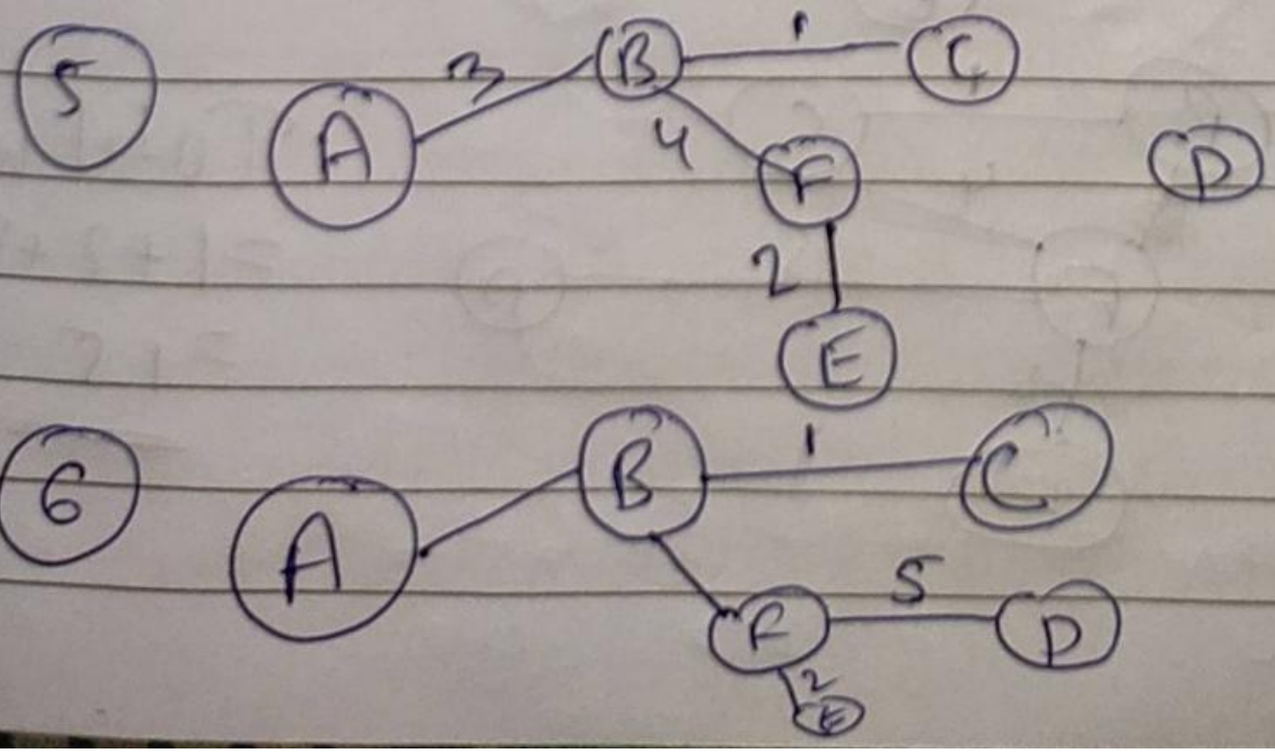
(3)



(4)



(5)

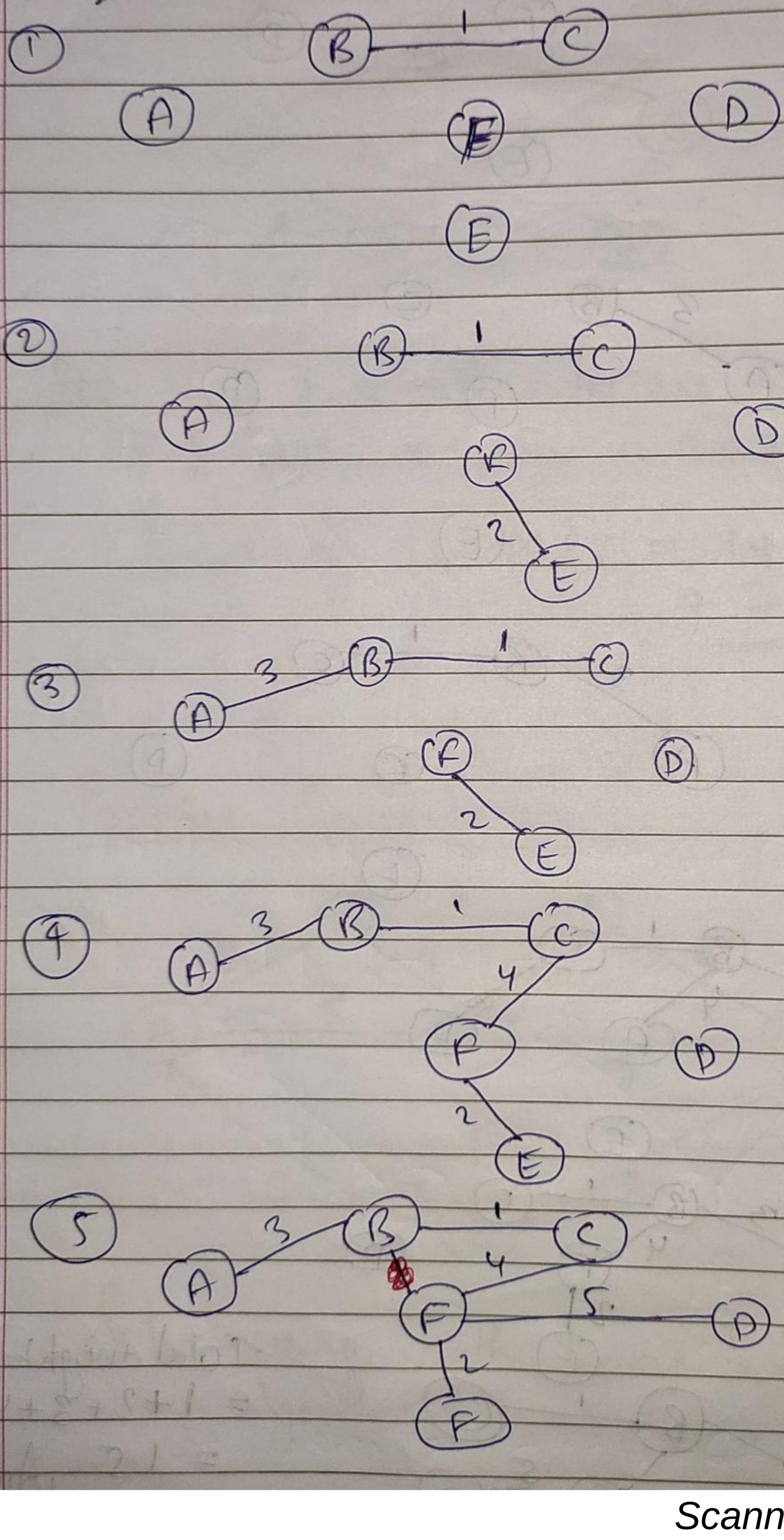


(6)

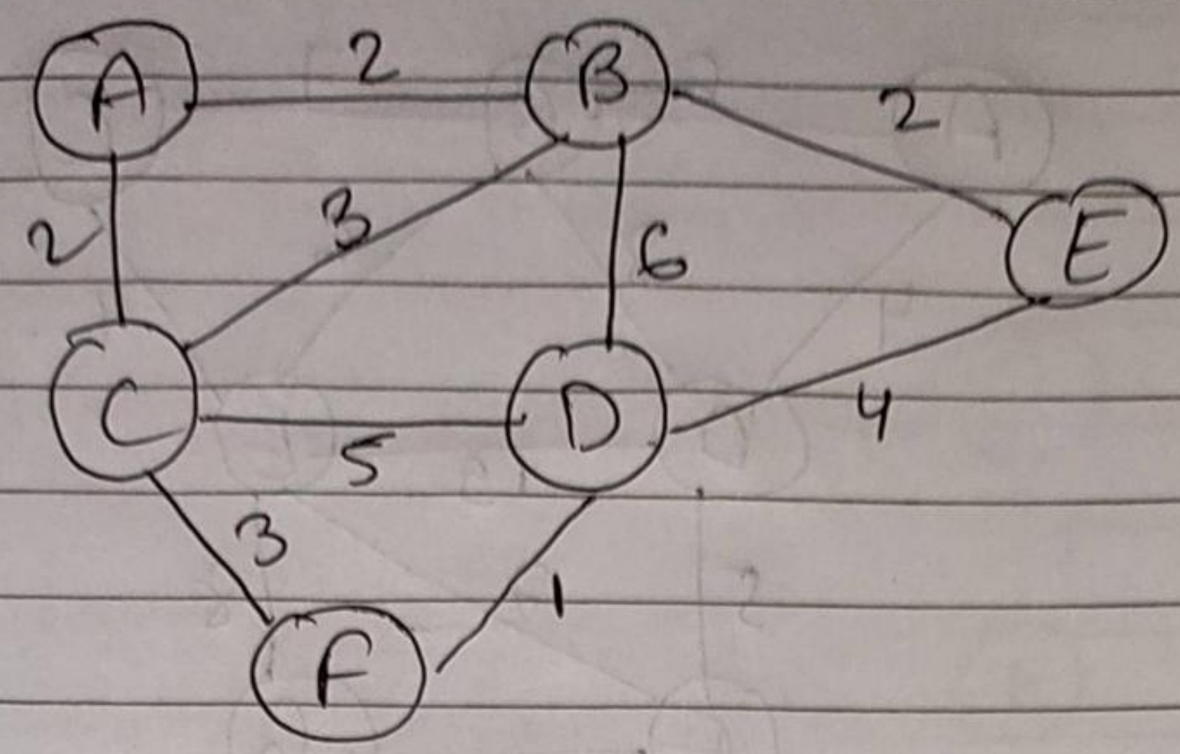
Total weight
= 1+2+3+4+5
= 15 Ans

Using Kruskal's :-

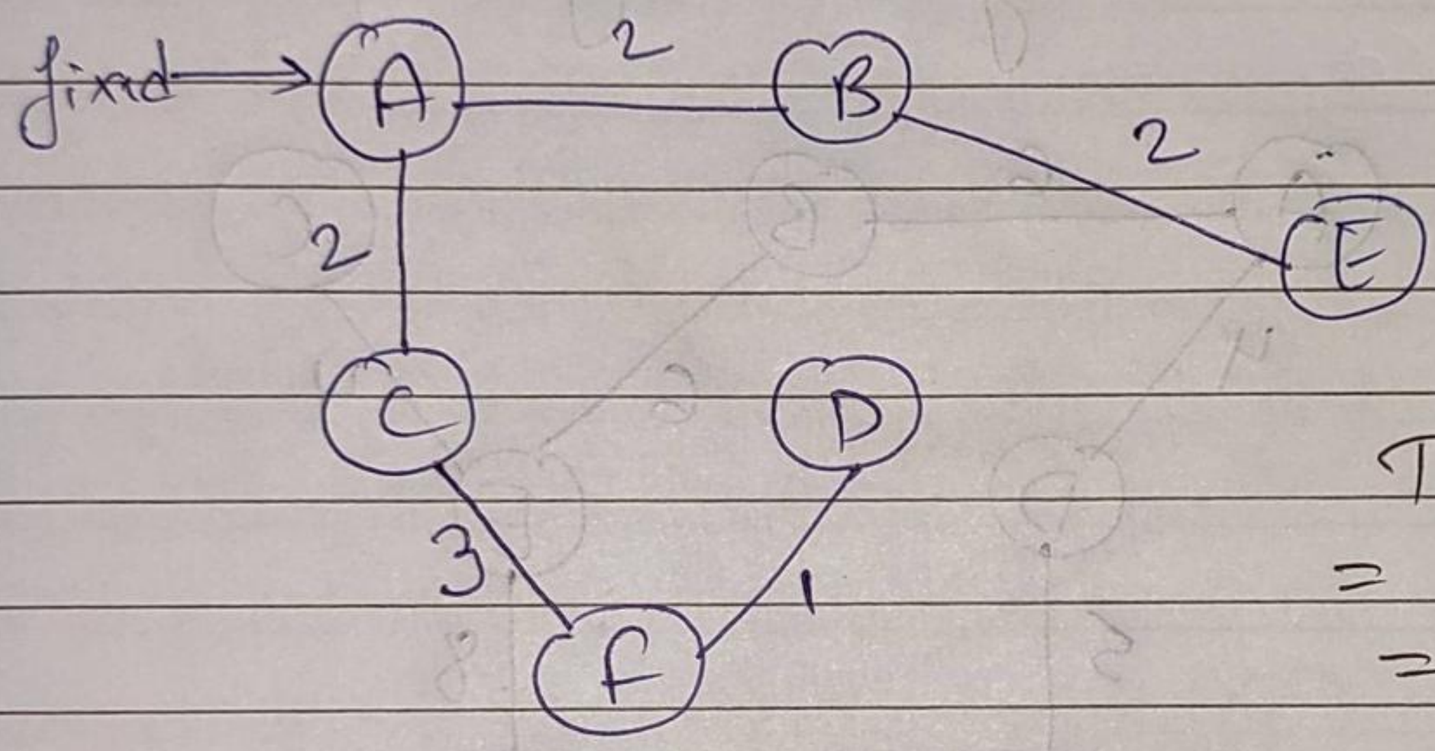
~~1~~ ~~2~~ ~~3~~ ~~4~~ 4 ~~5~~ 6 6 8



Q.2

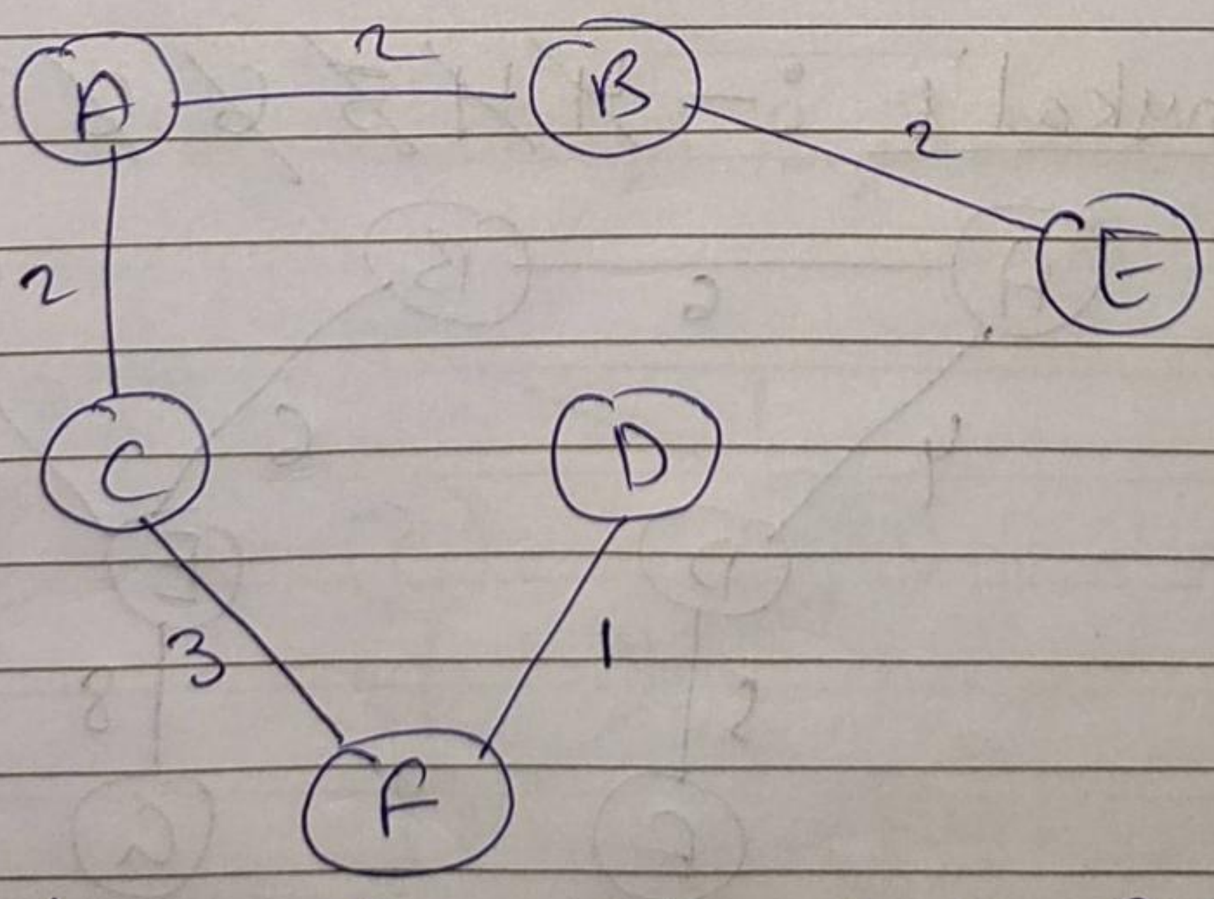


Sol → Prims :-



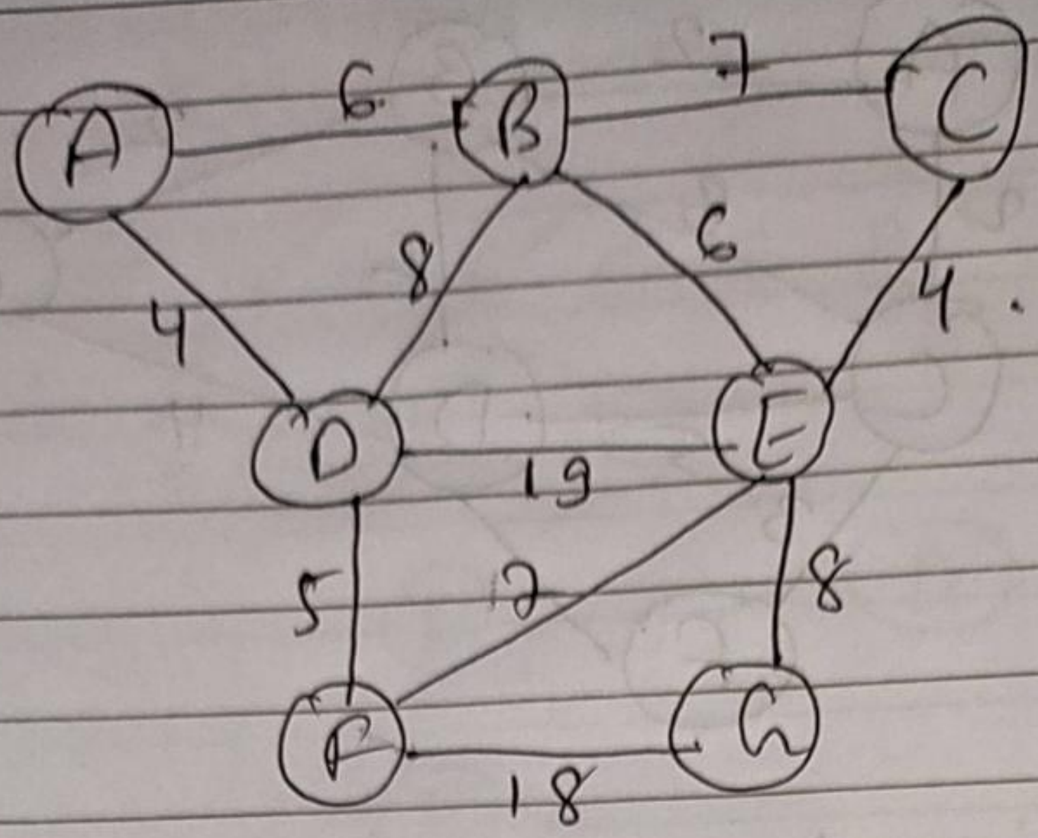
Total weight
= 1+2+2+2+3
= 10 Ans

Kruskal's :- ✗✗✗✗✗3456

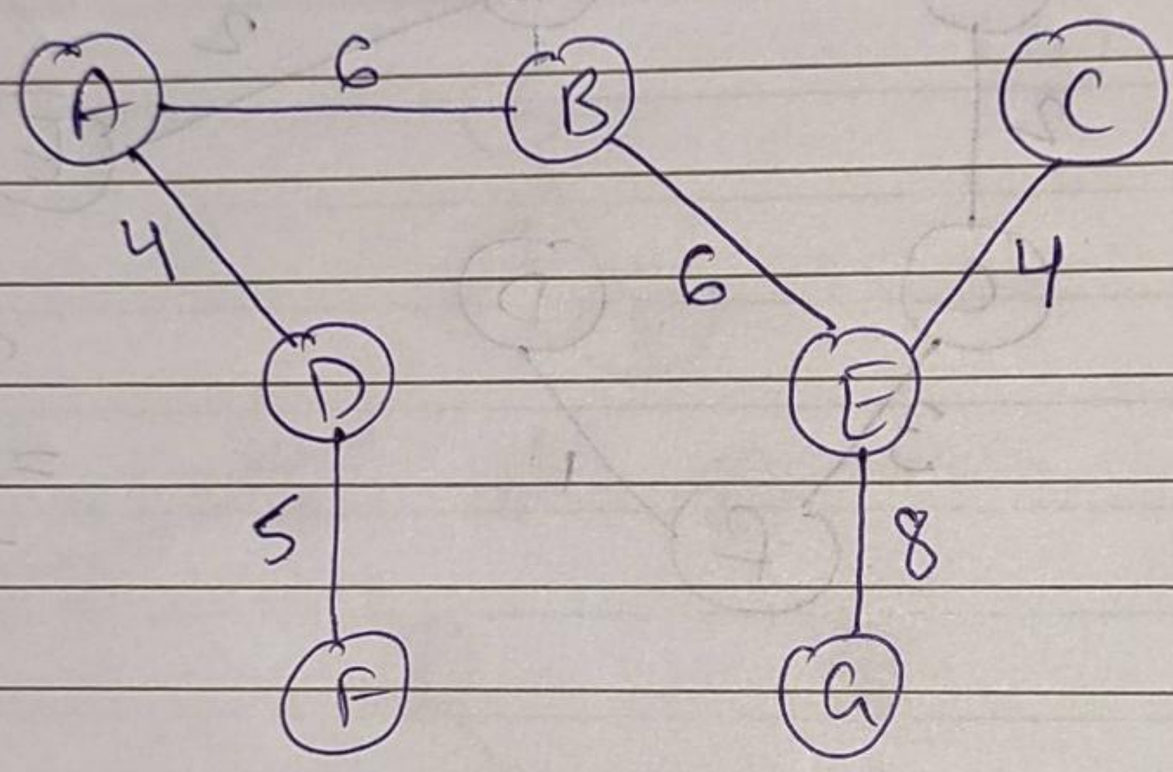


Total weight = 1+2+2+2+3
= 10 Ans

Q.

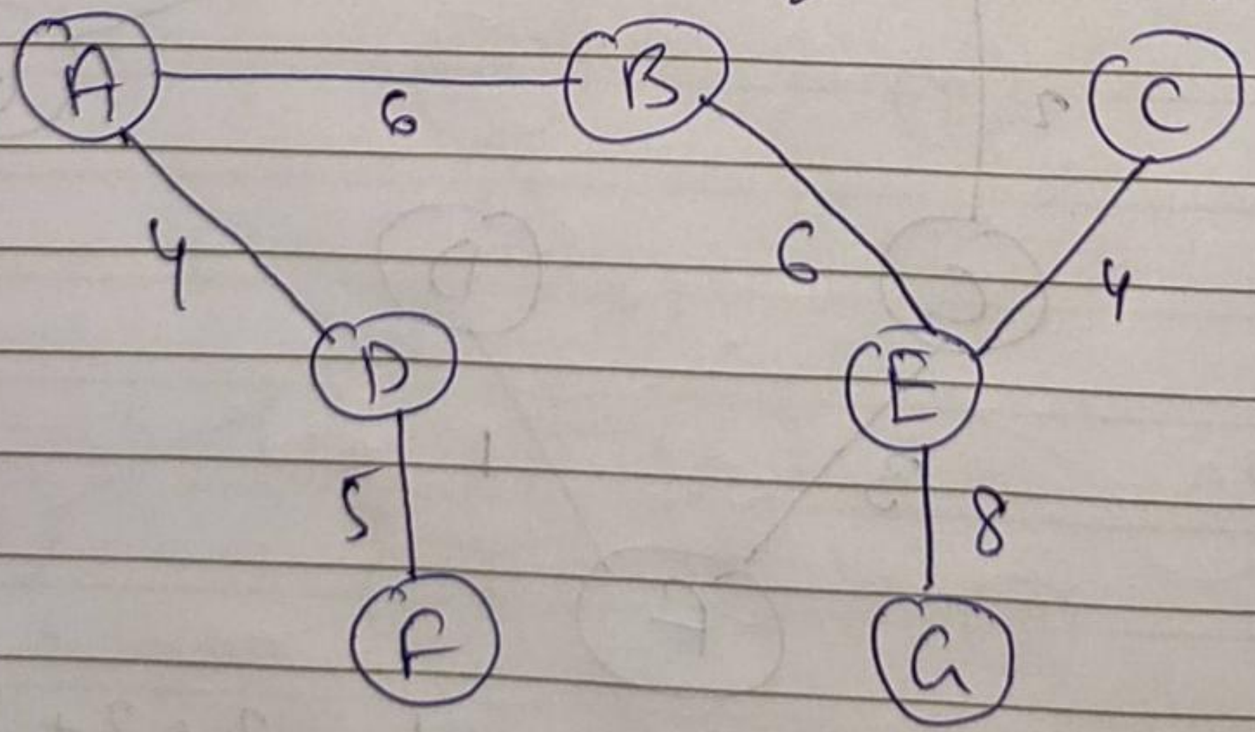


Sol → Prim's using D as fixed point.



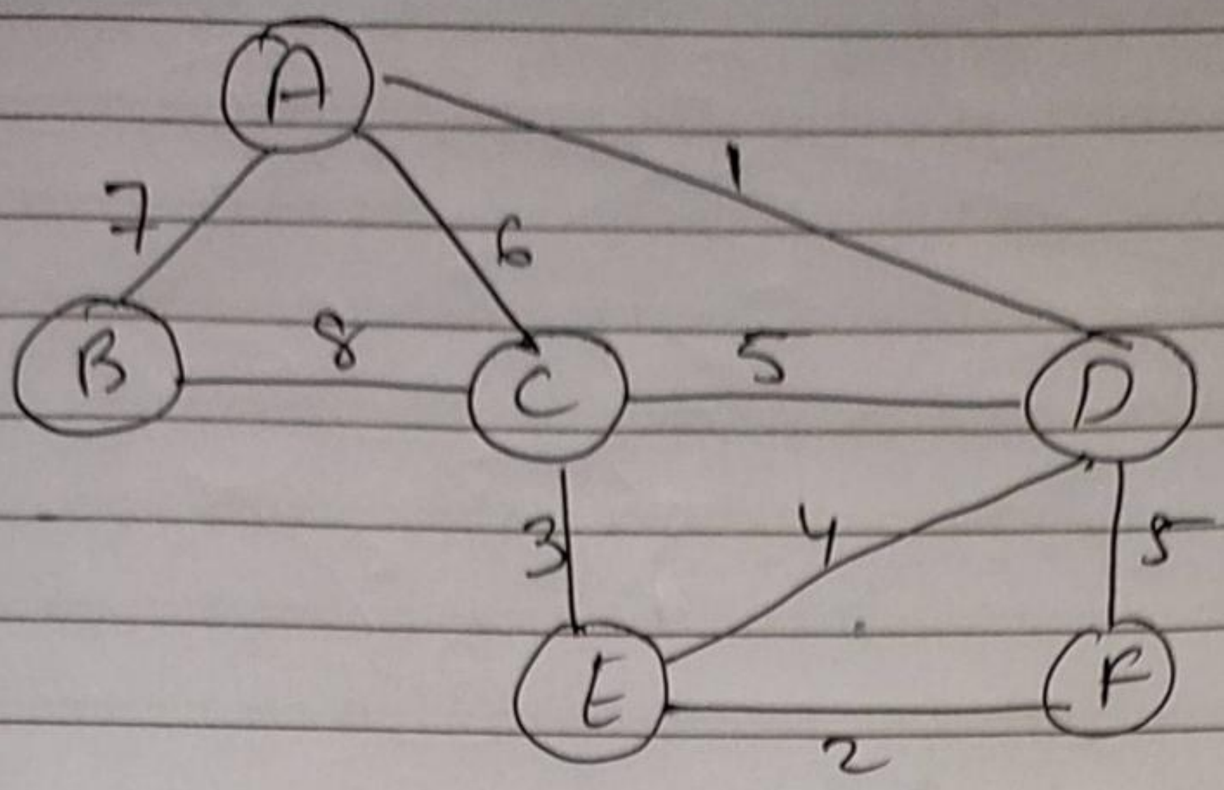
Total weight = 4+5+4+6+6+8 = 33.

Kruskal's :- ~~4~~ ~~4~~ ~~5~~ ~~6~~ ~~6~~ 7 7 ~~8~~ 8 18 19

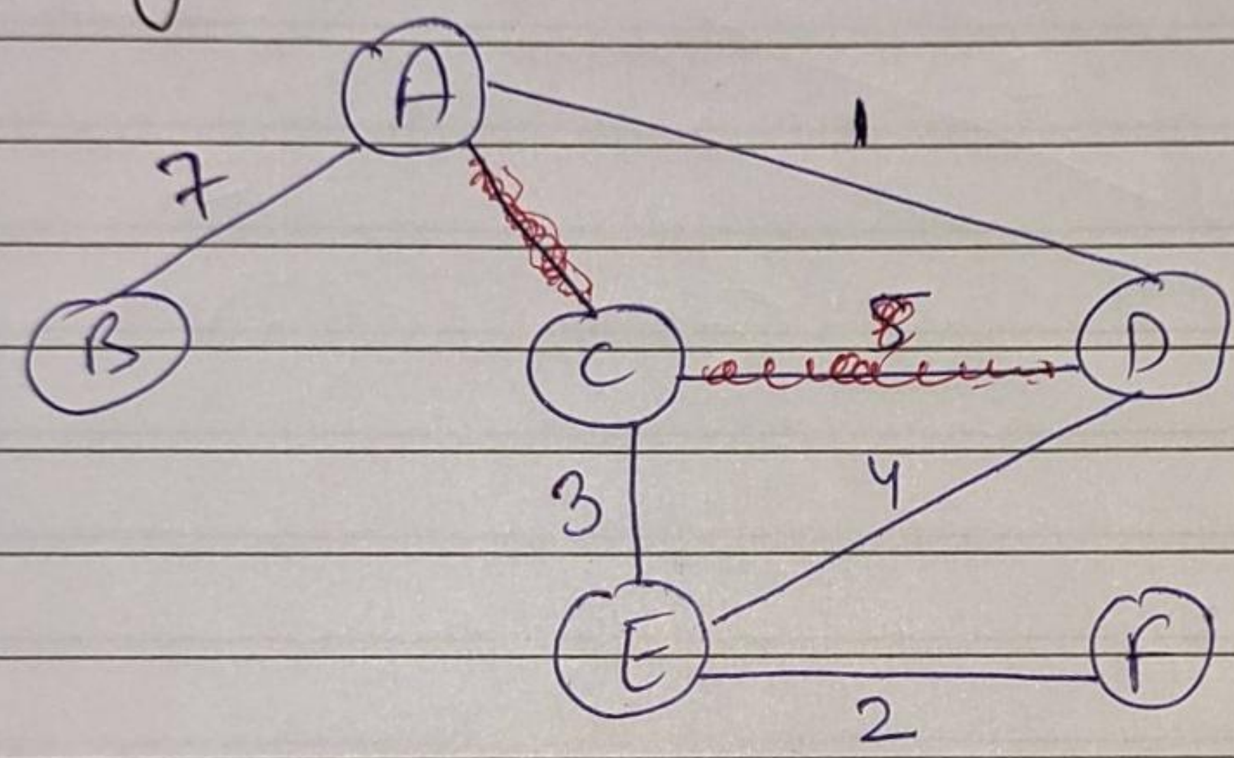


Total weight = 4+4+5+6+6+8
= 33 Ans

Q.

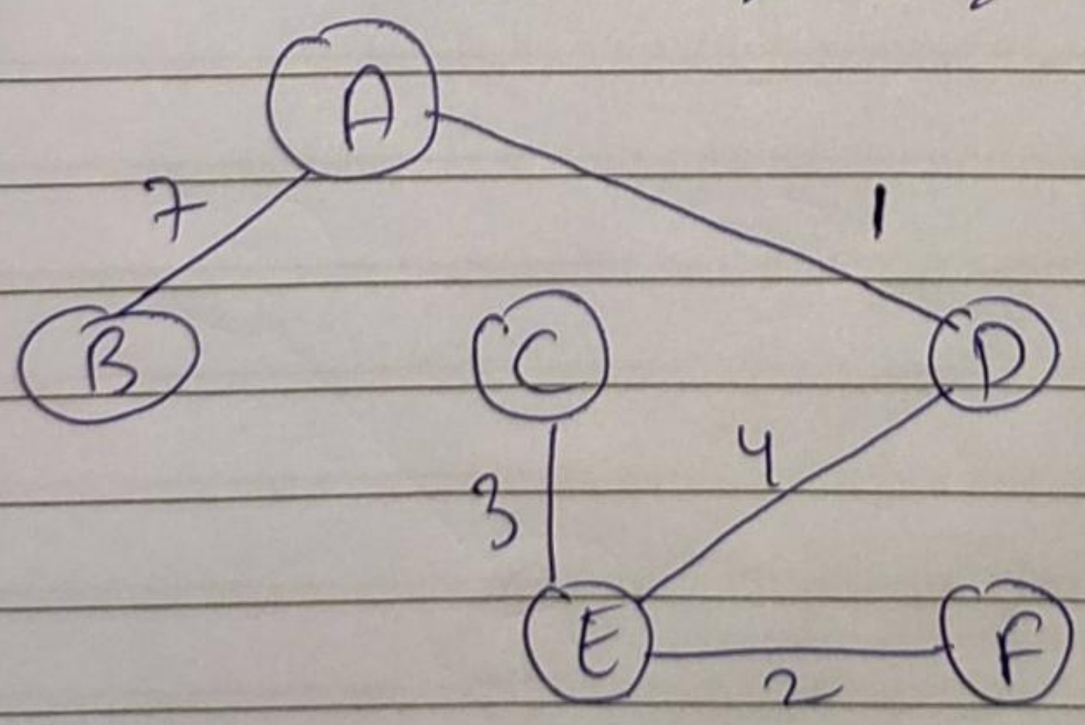


Sol → Using prim's :-



Total weight = 1+2+3+4+7 = 17 Ans

Using Kruskal's :- ~~1~~ 2 ~~3~~ ~~4~~ 5 5 6 ~~7~~ ~~8~~



Total weight = 1+2+3+4+7 = 17 Ans

Dynamic Programming

* It is similar to divide & conquer the difference is that in this we store the value of subproblem solution in a table to use it again in future. It follows Principle of optimality.
* Dynamic programming applies when the subproblem overlaps.

4 Steps :-

* Characterize the structure of an optimal solution.
* Recursively define the value of an optimal solution
* Compute the value of an optimal solution(in bottom up fashion.
* Construct optimal solution from computed information.

Examples :-

1) 0/1 Knapsack
2) Matrix chain Multiplication
3) Longest Common Subsequence
4) Optimal Binary Search Tree.

1) 0/1 Knapsack :-

It is like fractional knapsack. Difference b/w the two is that in this either we take entire or take nothing.

Q. Given knapsack capacity = 5 kg.

Item	weight	value
I_1	2	3
I_2	3	4
I_3	4	5
I_4	5	6

Ans →

weight →

Items ↓	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

weight of 2 & 3 = 5
value = 3 + 4 = 7

⇒ Maximum value = 7
Items I & II are involved.

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

Q.

Item	Weight	Values
I_1	2	1
I_2	3	2
I_3	4	5
I_4	5	6

Knapsack Capacity = 8.

Sol →

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

Maximum value = 8.

Items involved → 2 & 4.

Using formula :↓

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

$$V[4, 4] = \max\{V[4-1, 4], V[4-1, 4-w[4]] + P[4]\}.$$

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

$$= \max\{5, \text{0+6}\}$$

$$= 5$$

neglected (-ve)

Q for practise :-

1)

Item	Weight	Value
1	2	20
2	4	25
3	8	60

Knapsack capacity = 12 kg

2)

Item	-	-
1	2	3
2	3	7
3	4	2
4	5	9

Capacity = 5 kg

Longest Common Subsequence (LCS)

Longest Common Subsequence is a sequence of that appear in the same relative order but not necessarily contigous ~~that~~ (not substring) in both the string. i.e. element is not required to occupy consequtive position within orginal sequences.

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

find the LCS of these two string.

	y:	B	D	C	B
X:	0	0	0	0	0
B	0	↖1	←1	←1	↖1
A	0	↑1	↑1	↑1	↑1
C	0	↑1	↑1	←2	←2
D	0	↑1	←2	↑2	↑2
B	0	←1	↑2	↑2	←3

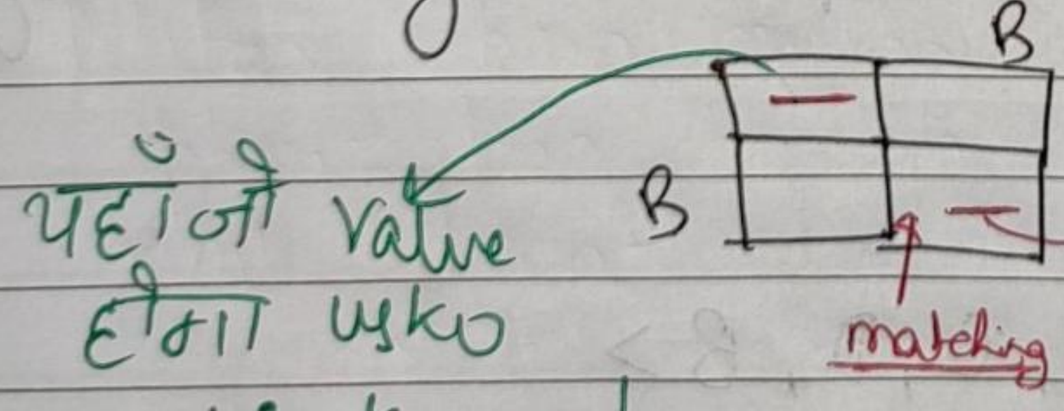
⇒ BCB Ans

how we select ↓

सबसे Last से Start करेंगे and वहाँ Arrow को Follow करेंगे।
Arrow '←' वाला होने पर Circle करेंगे।

How we are filling :-

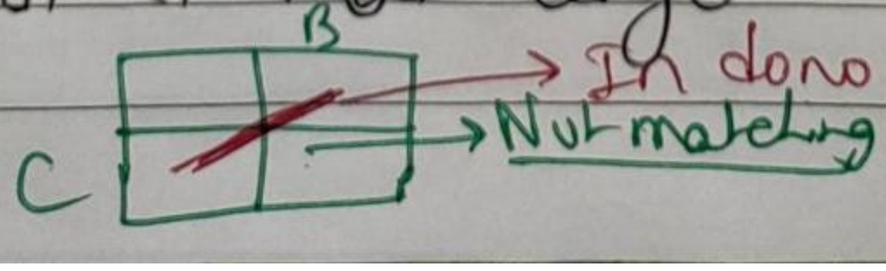
* Xi वाला को Yi के साथ Compare करेंगे, If matching. Then diagonal की value me +1 karenge.



पहाँ जो value होगी उसको +1 karenge

And yaha put karo us value के साथ

* Match nhi h to side & uppar वाला value में से Max lenge.



max value ko lenge.

Q. X = ⟨A, B, C, B, D, A, B⟩
Y = ⟨B, D, C, A, B, A⟩.

Ans →

	Yj:	(B)	D	(C)	A	(B)	(A)
Xi:	0	0	0	0	0	0	0
A	0	↑0	↑0	↑0	↖1	←1	←1
(B)	0	←1	←1	←1	↑1	←2	←2
(C)	0	↑1	↑1	←2	←2	↑2	↑2
(B)	0	←1	↑1	↑2	↑2	←3	←3
D	0	↑1	←2	↑2	↑2	↑3	↑3
(A)	0	↑1	↑2	↑2	←3	↑3	←4
B	0	←1	↑2	↑2	↑3	←4	↑4

⇒ BCBA are

Algorithm :-

```
LCS_LENGTH (x, y).
1.    m ← length [x]
2.    n ← length [y]
3.    for i ← 1 to m
4.      do c[i, 0] ← 0
5.    for j ← 1 to m
6.      do c[0, i] ← 0
7.    for i ← 1 to m
8.      do for j ← 1 to n
9.      do if xi = yj
10.   then c[i, j] ← c[i-1, j-1] + 1
11.     b[i,j] ← "↖"
```

```
12. else if c[i-1, j] ≥ c[j, j-1]
13.     then c[i,j] ← c[i-1, j]
14.          b[i,j] ← "↑"
15.     else c[i,j] ← c[j, j-1]
16.          b[i,j] ← "←"
17. return c & b.
```

```
PRINT_LCS (b, x, i, j).
1.  if i=0 or j=0
2.  then return
3.  if b[i,j] = "↖"
4.  then PRINT_LCS (b, x, i-1, j-1)
5.  print x;
6.  else if b[i,j] = "↑"
7.  then PRINT_LCS (b, x, i-1, j)
8.  else PRINT_LCS (b, x, i, j-1).
```

Matrix Chain Multiplication

Matrix chain multiplication is an optimization problem concerning the most efficient way to multiply a given sequence of matrices. The problem is not actually to perform multiplication but merely to decide the sequence of matrix multiplication involved.

Q Find the optimal parenthesis of a matrix multiplication whose sequence of dimensions is $\langle 5, 4, 6, 2, 7 \rangle$ i.e the matrix are as follows.

$A_1 (5 \times 4)$
$A_2 (4 \times 6)$
$A_3 (6 \times 2)$
$A_4 (2 \times 7)$.

Soln :- 5, 4, 6, 2, 7
$\Rightarrow$ 5 elements
$\therefore$ No. of blocks = 5 - 1 = 4. (i <= 1 to 4)

1	2	3	4	
0	120	88	158	1
	0	48	104	2
		0	84	3
			0	4

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

5 4 6 2 7
↓ ↓ ↓ ↓ ↓
P_0 P_1 P_2 P_3 P_4

$$m[i,j] = \min\{ m[i,k] + m(k+1, j) + P_{i-1} P_k P_j \}$$
where $i \le k < j$

$\therefore m[1,2] = \min[m[1,1] + m[2,2] + P_0 P_1 P_2]$
$= \min\{ 0 + 0 + 5 \cdot 4 \cdot 6 \}$
$= 120$

$$m[2,3] = \min\left[m[2,2] + m[3,3] + P_1 P_2 P_3\right]$$

$$= \min(0 + 0 + 4 \cdot 6 \cdot 2)$$

$$= 48$$

$$m[3,4] = \min\left[m[3,3] + m[4,4] + P_2 P_3 P_4\right]$$

$$= \min(0 + 0 + 6 \cdot 2 \cdot 7)$$

$$= 84$$

$$m[1,3] = \min\begin{cases} m[1,1] + m[2,3] + P_0 \cdot P_1 \cdot P_3 \\ m[1,2] + m[3,3] + P_0 \cdot P_2 \cdot P_3 \end{cases}$$

$$= \min\begin{cases} 0 + 48 + 5 \cdot 4 \cdot 2 \\ 120 + 0 + 5 \times 6 \times 2 \end{cases}$$

$$= \min(88, 180)$$

$$= 88 \quad \text{with } k = 1$$

$$m[2,4] = \min\begin{cases} m[2,2] + m[3,4] + P_1 P_2 P_4 \\ m[2,3] + m[4,4] + P_1 P_3 P_4 \end{cases}$$

$$= \min\begin{cases} 0 + 84 + 4 \cdot 6 \cdot 7 \\ 48 + 0 + 4 \cdot 2 \cdot 7 \end{cases}$$

$$= \min(252, 104)$$

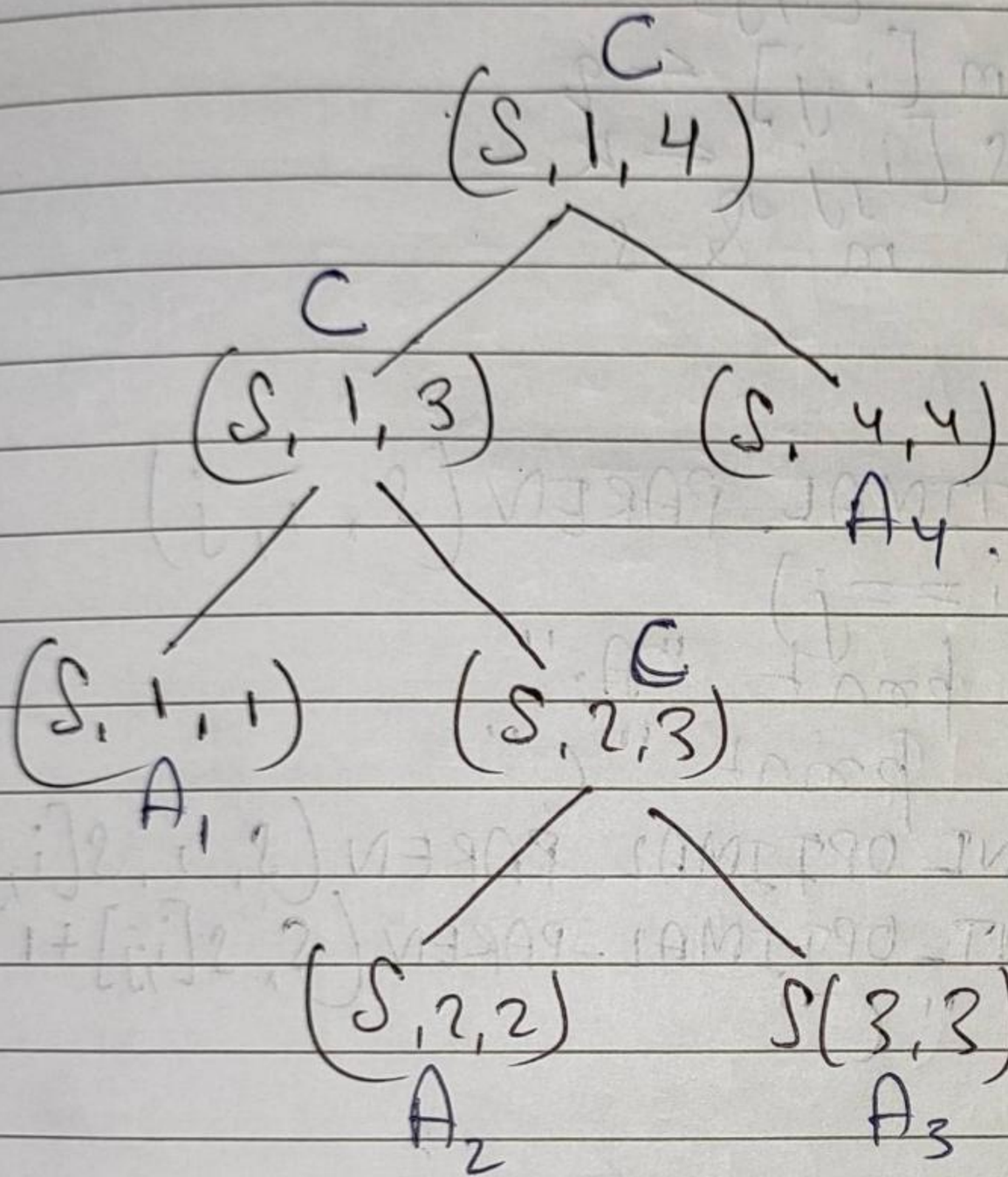
$$= 104 \quad \text{with } k = 3$$

$$m[1,4] = \min\begin{cases} m[1,1] + m[2,4] + P_0 P_1 P_4 \\ m[1,2] + m[3,4] + P_0 P_2 P_4 \\ m[1,3] + m[4,4] + P_0 P_3 P_4 \end{cases}$$

$$= \min\begin{cases} 0 + 104 + 5 \cdot 4 \cdot 7 \\ 120 + 84 + 5 \cdot 6 \cdot 7 \\ 88 + 0 + 5 \cdot 2 \cdot 7 \end{cases}$$

$= \min(244, 414, 158)$

$= 158$ with $k = 3$.



$\Rightarrow ((A_1) \cdot (A_2 A_3) \cdot A_4)$. Ans

Algorithm :-

1. $n \leftarrow$ length$[p] - 1$
2. for $i = 1$ to n
3. do $m[i, j] \leftarrow 0$
4. for $l \leftarrow 2$ to n
5. do for $i \leftarrow 1$ to $n - l + 1$

```
6.    do j ← i + l - 1
7.       m[i, j] ← ∞
8.       for k ← 1 to j - 1
9.       do q ← m[i, k] + m[k+1, j] + P_{i-1} P_k P_j
10.      if q < m[i, j]
11.      then m[i, j] ← q
12.           s[i, j] ← k
13.   return m & s
```

```
PRINT_OPTIMAL_PAREN(S, i, j)
1.    if (i == j)
2.       then print "A_i"
3.       else print "("
4.       PRINT_OPTIMAL_PAREN(S, i, s[i, j])
5.       PRINT_OPTIMAL-PAREN(S, s[i, j] + 1, j)
6.       print ")"
```

Optimal Binary Search Tree

Binary Search Tree :- Tree in which left subtree contain nodes with keys lesser than node's key and right subtree with node greater than node's key.

Q

	1	2	3	4
Keys	10	20	30	40
frequency	4	2	6	3

Ans →

0	1	2	3	4	
0	4	8^1	20^3	26^3	0
	0	2	10^3	16^3	1
		0	6	12^3	2
			0	3	3
				0	4

← ∞ or 7

$$c[i,j] = \min_{i \le k < j} \{ c[i,k] + c[k+1,j] + w[i,j] \}$$

$$c[0,1] = \min[c[0,0] + c[1,1] + w[0,1]]$$
$$= \min(0+0+4)$$
$$= 4.$$

$$c[1,2] = \min[c[1,1] + c[2,2] + w[1,2]]$$
$$= \min(0+0+2)$$
$$= 2$$

$$c[2,3] = \min[c[2,2] + c[3,3] + w[2,3]]$$
$$= \min[0+0+6]$$
$$= 6$$

$$c[3,4] = \min[c[3,3] + c[4,4] + w[3,4]]$$
$$= \min(0+0+3)$$
$$= 3$$

$$c[0,2] = \min \begin{cases} c[0,0] + c[1,2] + w[0,2] \\ c[0,1] + c[2,2] + w[0,2] \end{cases}$$

$$= \min \begin{cases} 0 + 2 + (4+2) \\ 4 + 0 + (4+2) \end{cases}$$

$$= \min(8, 10)$$

$$= 8 \quad \text{with} \quad k+1 = 1$$

$$c[1,3] = \min \begin{cases} c[1,1] + c[2,3] + w[1,3] \\ c[1,2] + c[3,3] + w[1,3] \end{cases}$$

$$= \min \begin{cases} 0 + 6 + (2+6) \\ 2 + 0 + 8 \end{cases}$$

$$= \min(14, 10)$$

$$= 10 \quad \text{with} \quad k+1 = 3.$$

$$c[2,4] = \min \begin{cases} c[2,2] + c[3,4] + w[2,4] \\ c[2,3] + c[4,4] + w[2,4] \end{cases}$$

$$= \min \begin{cases} 0 + 3 + 9 \\ 6 + 0 + 9 \end{cases}$$

$$= \min(12, 15)$$

$$= 12 \quad \text{with} \quad k+1 = 3$$

$$c[0,3] = \min \begin{cases} c[0,0] + c[1,3] + w[0,3] \\ c[0,1] + c[2,3] + w[0,3] \\ c[0,2] + c[3,3] + w[0,3] \end{cases}$$

$$= \min \begin{cases} 0 + 10 + 12 \\ 4 + 6 + 12 \\ 8 + 0 + 12 \end{cases}$$

$$= \min(22, 22, 20)$$

$$= 20 \quad \text{with} \quad k+1 = 3$$

$$c[1,4] = \min \begin{cases} c[1,1] + c[2,4] + w[1,4] \\ c[1,2] + c[3,4] + w[1,4] \\ c[1,3] + c[4,4] + w[1,4] \end{cases}$$

$$= \min \begin{cases} 0 + 12 + 11 \\ 2 + 3 + 11 \\ 10 + 0 + 11 \end{cases}$$

$$= \min\{23, 16, 21\}$$

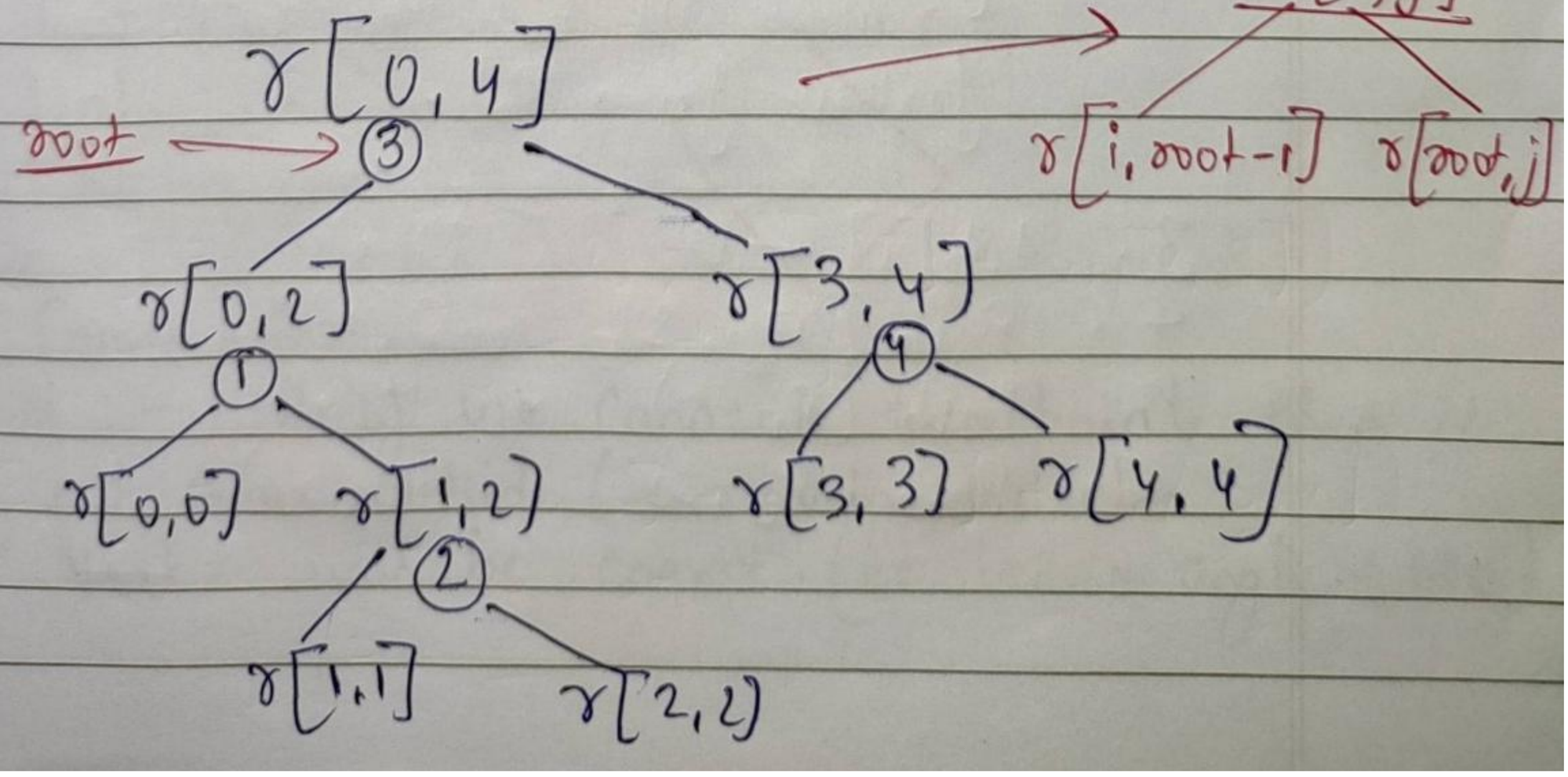
$$= 16 \quad \text{with } k+1 = 3.$$

$$c[0,4] = \min \begin{cases} c[0,0] + c[1,4] + w[0,4] \\ c[0,1] + c[2,4] + w[0,4] \\ c[0,2] + c[3,4] + w[0,4] \\ c[0,3] + c[4,4] + w[0,4] \end{cases}$$

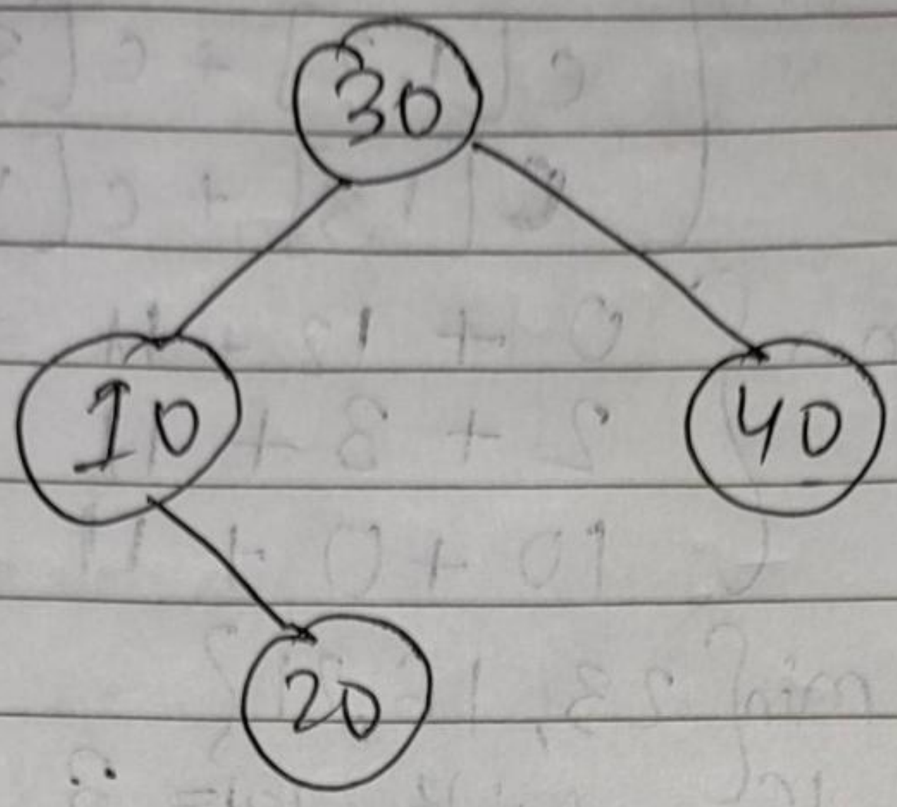
$$= \min \begin{cases} 0 + 16 + 15 \\ 4 + 12 + 15 \\ 8 + 3 + 15 \\ 20 + 0 + 15 \end{cases}$$

$$= \min\{31, 31, 26, 35\}$$

$$= 31 \quad \text{with } k+1 = 3$$



⇒



Ans