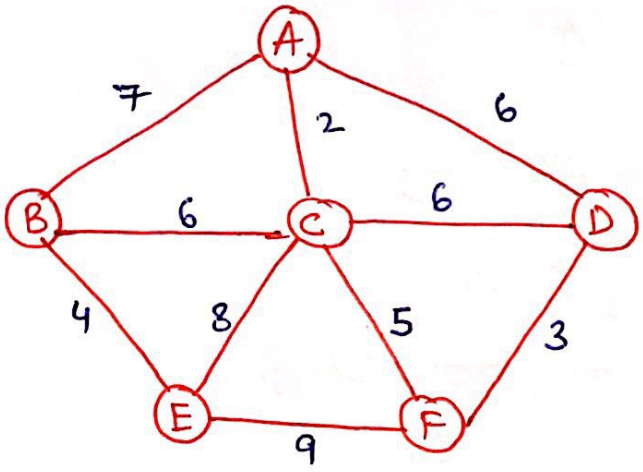
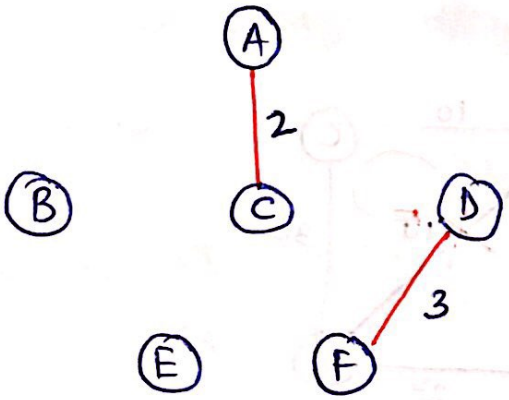
eg:-

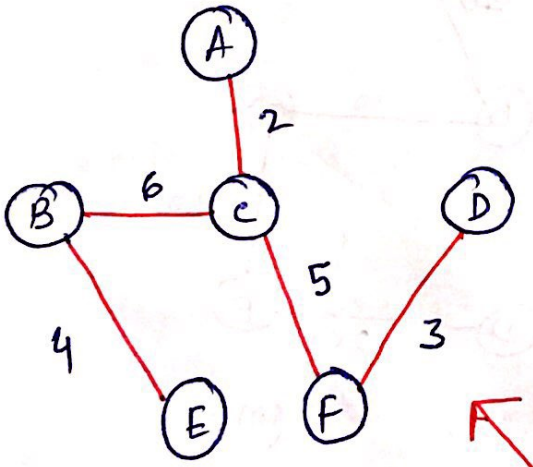


Apply Kruskal & prims algorithm ?

Kruskal's algorithm :-



AC DF B E



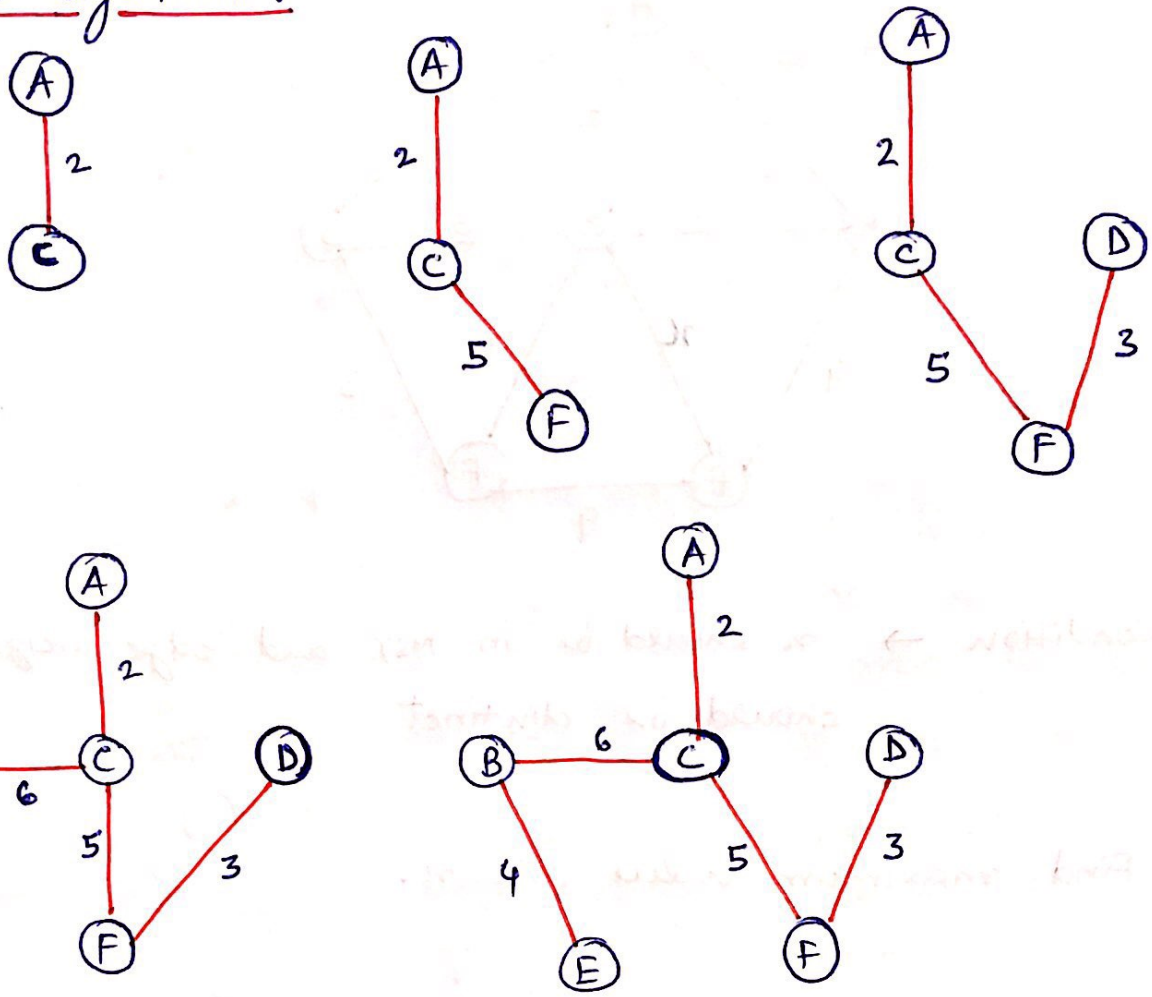
AC DF BE

ACDF BF

ABCDEF

cost = 20

Prims algorithm :-



cost = 20

eg:- what is prims algo seqmence of edges in above question ?

(i) ✓ (A,C) (C,F) (F,D) (B,C) (B,E)

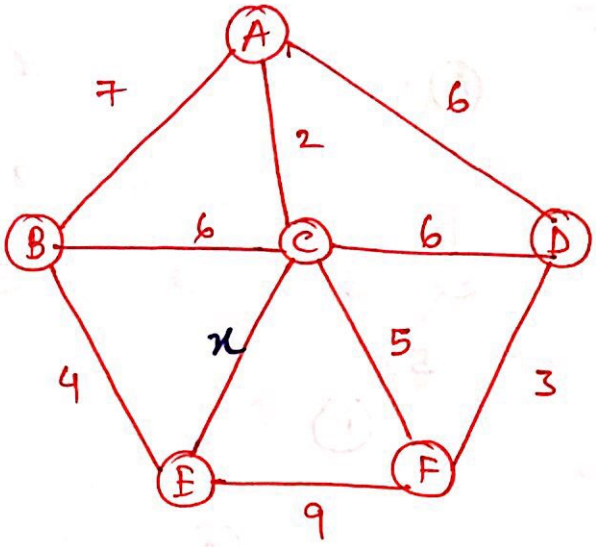
(ii) (A,C) (C,F) (B,E) (B,C) (F,D)

(iii) (E,B) (B,C) (C,A) (C,F) (F,D)

(iv) (E,B) (B,C) (F,D) (C,F) (C,A)

check for adjacency and connetedness

Q.

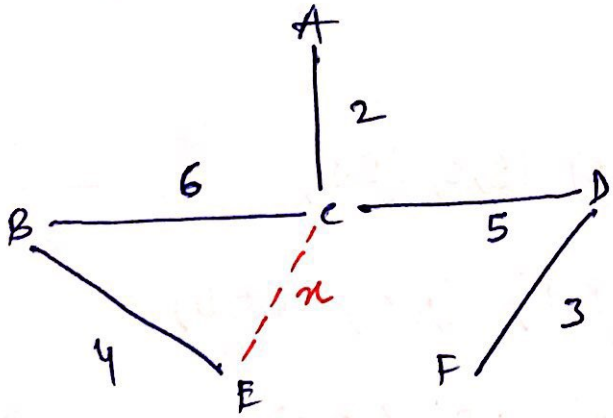


Condition → x should be in MST and edge weight should be distinct

Find maximum value of x.

Steps :-

(i) First make MST without considering x.



(ii) Now add x in MST.

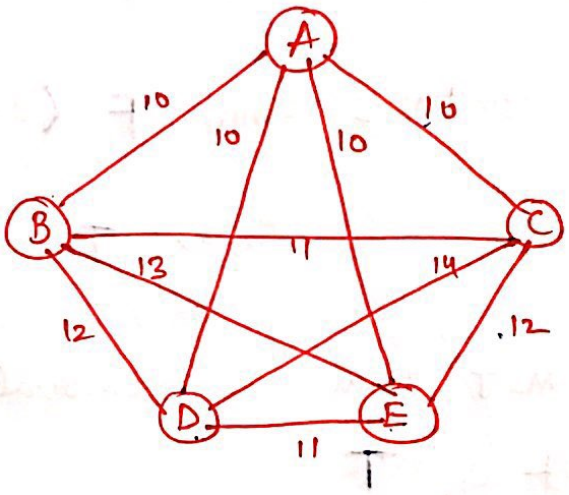
For add x maximum weight edge should be eliminated i.e $(x < 6)$

But, weight should be dintinct so,

$x \neq 5, 4, 3, 2$

so, $(x_{max} = 1)$

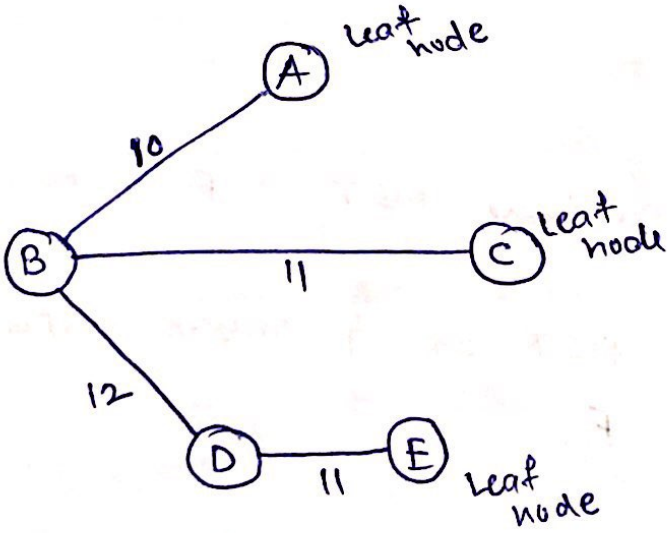
Q.



what is the cost of MST for above graph in such a way that 'A' is leaf node in MST.

Leaf node → Directed graph $(d_{out} = 0)$

→ Undirected graph $(d = 1)$



cost = 44

Q. Let G be an undirected weighted with distinct edge weight. E_{max} be the max edge weight and E_{min} be the min edge weight.

Check True and False.

(a) G – contain unique MST T (distinct edges)

(b) MST of G may contains e_{min} F (starting edge)

(c) " " e_{max} T

(d) if E_{max} is in MST then its removal from G must disconnect G. T

Q. Let G be an undirected weighted graph with n-vertices, w be the minimum edge weight among all edge, "e" be a specific edge with weight w.

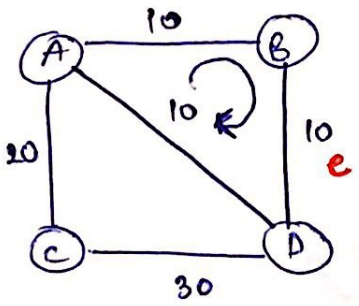
True / False:

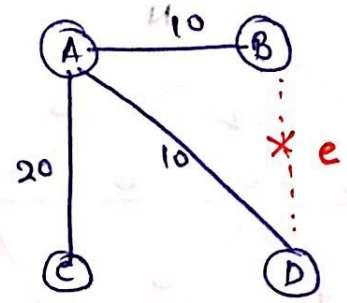
(a) G contain unique MST F (no-distinct edges)

(b) Every MST of G must contain contain e F

(c) Every MST of G must contain atleast 1-edge with weight w. T

(d) If e is not in MST, then in that cycle all edges contain same weight W. T





Q. Let G be an undirected unweighted connected graph with n vertices shown below by an "nxn" adjacency matrix in which

(i) All diagonal element 0's

(ii) All non-diagonal element is 1's.

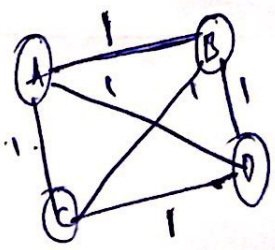
T/F ?

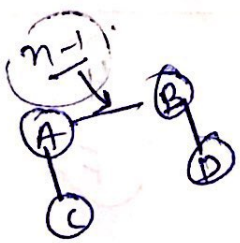
(a) G contain uniqe MST F

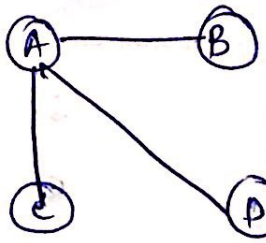
(b) G don't have MST F

(c) G contain multiple MST with different cost. F

(d) G contain multiple MST with same cost. T



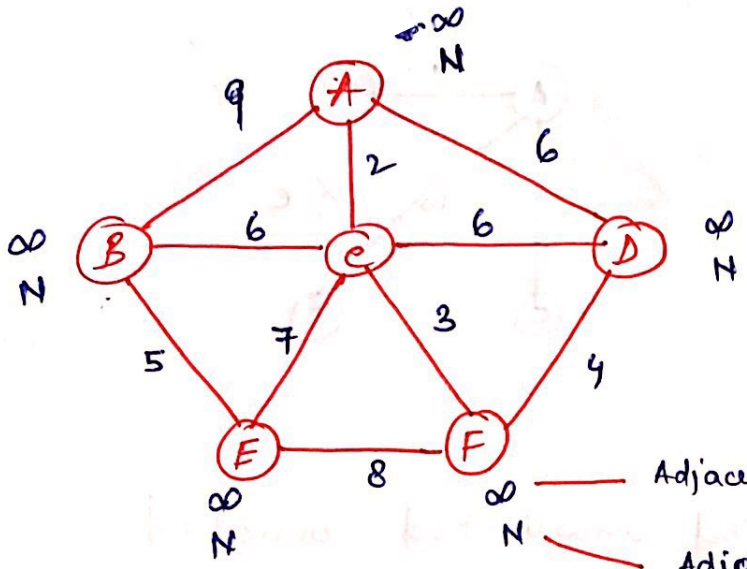




=3

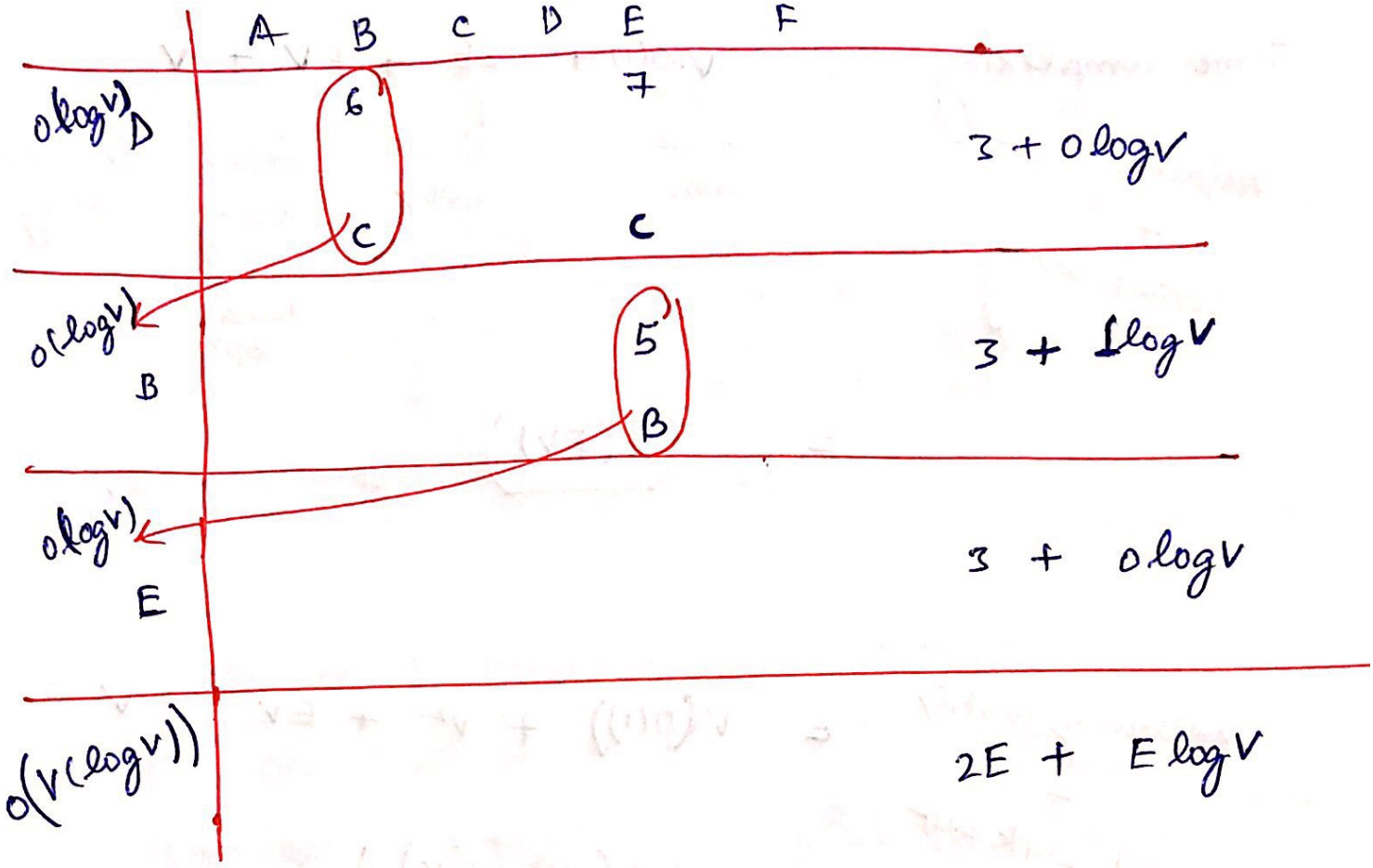
same cost = n-1 = 4-1

Time complexity of Prims algorithm :-



Build Minheap

A	B	C	D	E	F	
∞	∞	∞	∞	∞	∞	— O(V)
N	N	N	N	N	N	
choose one vertex (Let A)						
0	∞	∞	∞	∞	∞	— O(1)
N	N	N	N	N	N	
delete	Adjacent to A (decrease key opn)					
O(logV) A	9	2	6	∞	∞	3 + 3 log V
	A	A	A	N	N	
O(logV) C	6		6	7	3	5 + 4 log V
	C		A	C	C	
O(logV) F	6		4	7		3 + 2 log V
	C		F	C		



Time complexity = $V \log V + E + E \log V + V$

$= O((V+E) \log V)$ Adjacency List + Minheap

$= V \log V + V^2 + E \log V + V$

$= O(V^2 + E \log V)$ Adjacency matrix + Minheap

Time complexity = $V \cdot O(1) + 2E + EV + V$

- $V \cdot O(1)$: access min
- $2E$: adjacent vertex
- EV : adjust array after decrease opn
- V : initialize array

Adjacency List + sorted array

$= O(EV)$

Adjacency matrix + sorted Link List

$= V(O(1)) + V^2 + EV + V$

$= O(V^2 + EV)$

Adjacent List + unsorted array

$= V^2 + 2E + E + V$

$= O(V^2)$

- V^2 : perform selection sort pass for each vertex. $(V \times V)$
- E : no adjustment required (unsorted) for decrease opn.

Huffman Coding:-

- Non-uniform coding
- More freaueney characters ⇒ less bits.
- Data Compression technique

Message = 100 characters

a = 6%	b = 30%
c = 45%	d = 3%
e = 1%	f = 15%

Huffman coding

c — 45% — 1-bit
b — 30% — 2-bit
f — 15% — 3-bit
} major contribution

a — 6%
d — 3%
e — 1%

without compression

ASCII — 1 char — 1 byte (8-bits)

e — 60 × 8
b — 30 × 8
f — 15 × 8
a — 6 × 8
d — 3 × 8
e — 1 × 8

normal compression

Total characters = 6
$\lceil \log_2 6 \rceil$

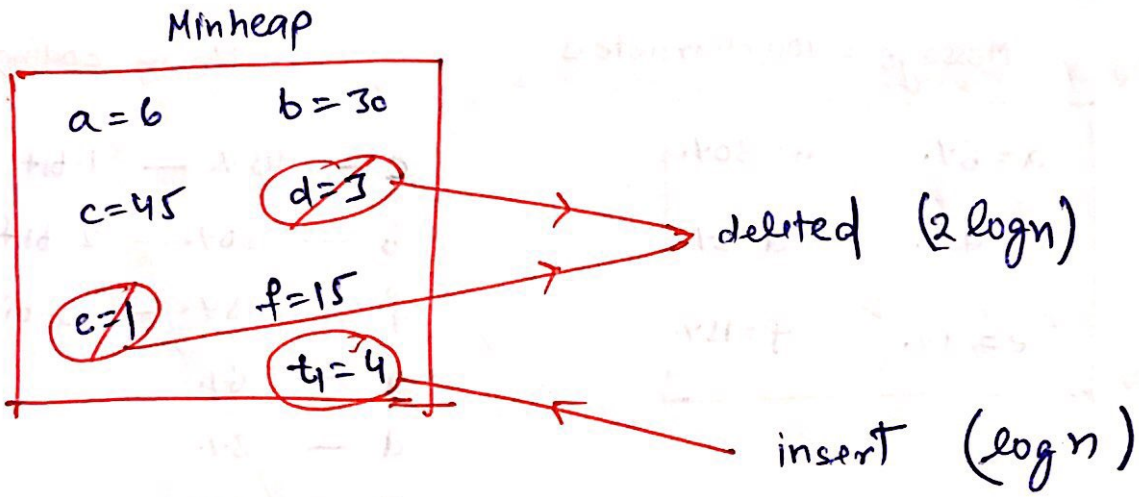
c — 45 × 3
b — 30 × 3
f — 15 × 3
a — 6 × 3
d — 3 × 3
e — 1 × 3

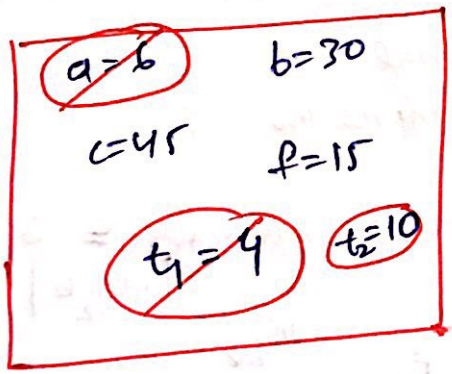
Algorithm :-

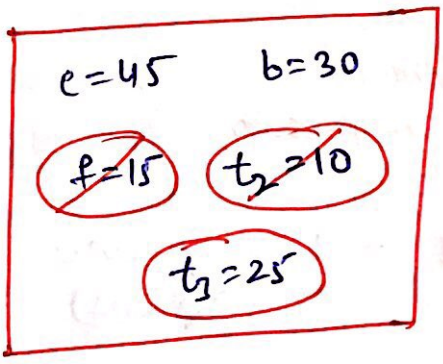
(i) Deletion 2 minimums from character space.

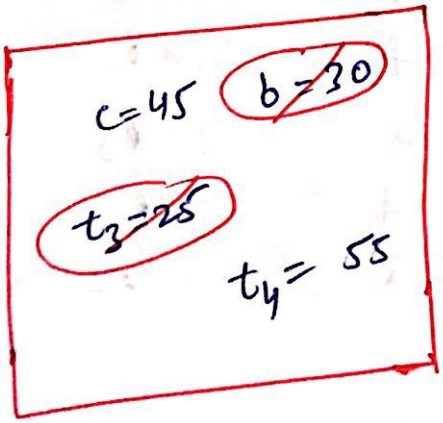
(ii) Insert sum of 2 deleted characters freaency into character space.

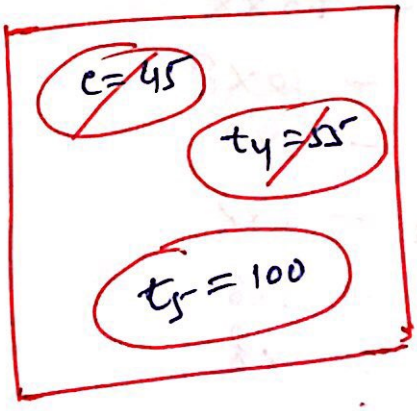
(iii) Repeat until single character is left in space.

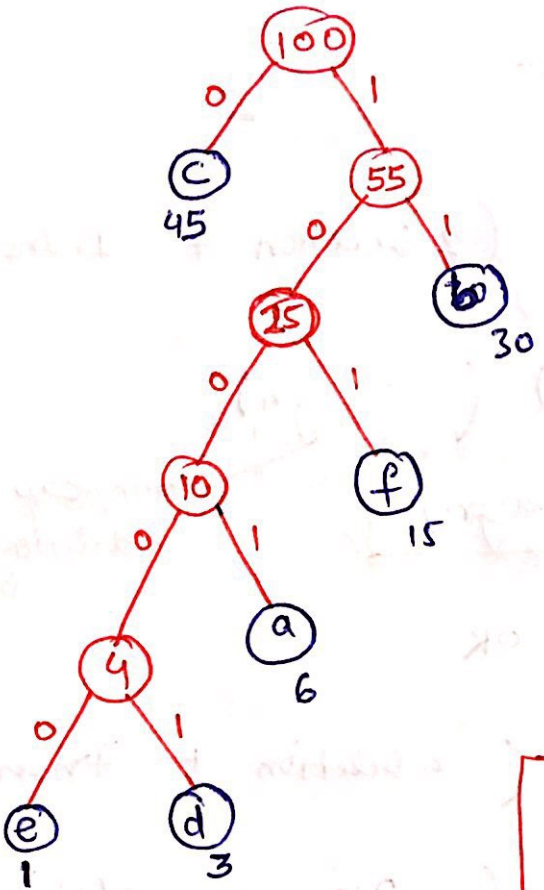




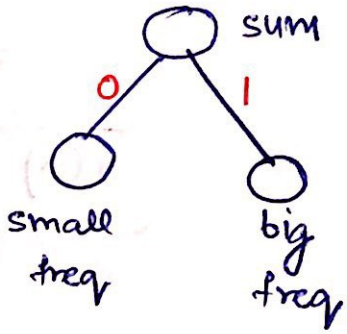








L = 0
R = 1



2-way huffman coding tree

Huffman code :-

e = 10000
d = 10001
a = 1001
f = 101
b = 11
c = 0

Total bits = $6 \times 4 + 30 \times 2 + 45 \times 1 + 3 \times 5 + 1 \times 5 + 15 \times 3$

$= 194$

Avg = $194/100$ = 1.94 bits/char

NOTE :-

- For K- way huffman coding tree, perform K deletion.
- More freq → less bits → (Minimum deleted)

More freq → more bits → (Maximum deleted)

Time - complexity :-

$T(n) = n-1\ (2\ \text{Deletion} + 1\ \text{Insertion}) + n$ (create minheap)

$= (n-1)\ (3\log n)$ (minheap deletion & insertion)

$T(n) = O(n \log n)$

OR

$T(n) = n-1\ (2\ \text{Deletion} + \text{Insertion}) + n$ (intializ array)

$= n-1\ (2n + O(1))$ (array deletion/insertion)

$T(n) = O(n^2)$

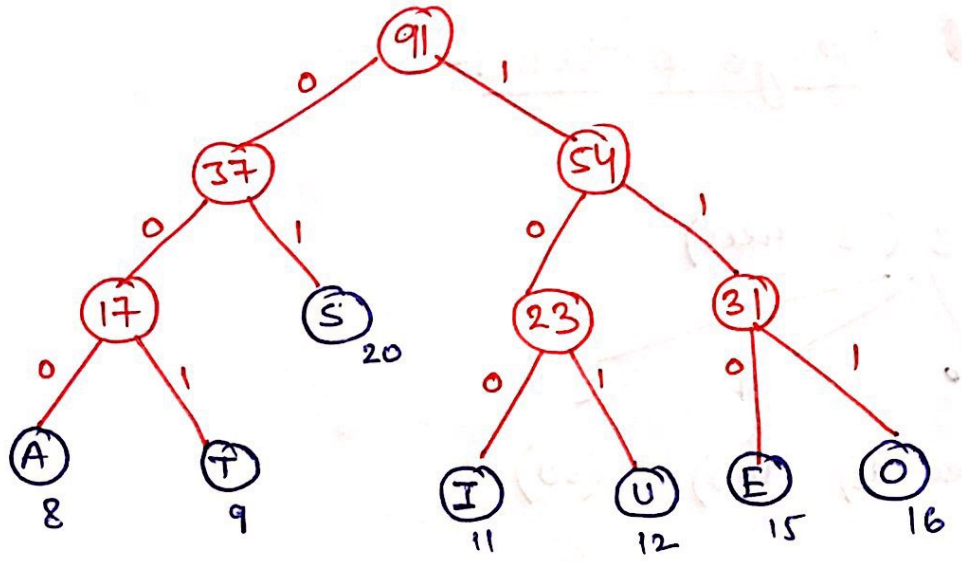
Q.

	A	E	I	O	U	S	T
M =							
f(M) =	8	15	11	16	12	20	9

Total characters = 91

Huffman code :-

A	000	O	111	T	001
E	110	U	101		
I	100	S	01		



Total bits = 253 , Avg/char = 253/91 bits/char

Encoded msg = 110 111 0000 10 10 110 1111 0001

E O A S T S U O I S

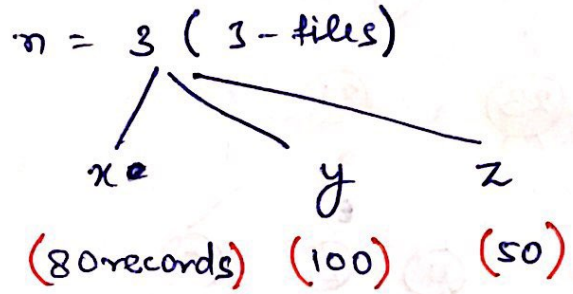
Decoded msg = (EOASTSUOIS)

NOTE:- Huffman coding also known as "optimal Prefix Notation" i.e no one code will not be prefix of some other codes.

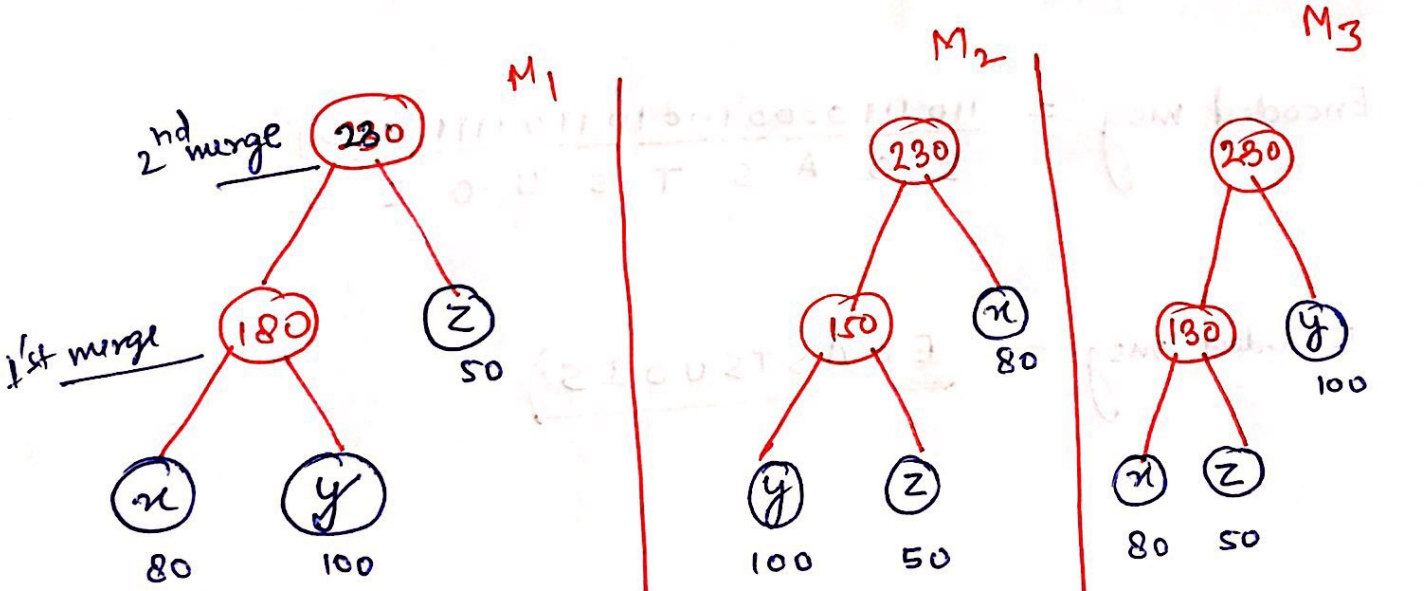
So no data damage.

Optimal Merge pattern:-

eg:-



Since all records are sorted so when we merge them we have to apply merge algorithm to make one sorted file.



Total move op^n = 180 + 230
= 410

Total mov op^n = 150 + 230
= 380

Total mov op^n
= 130 + 230
= 360

BEST

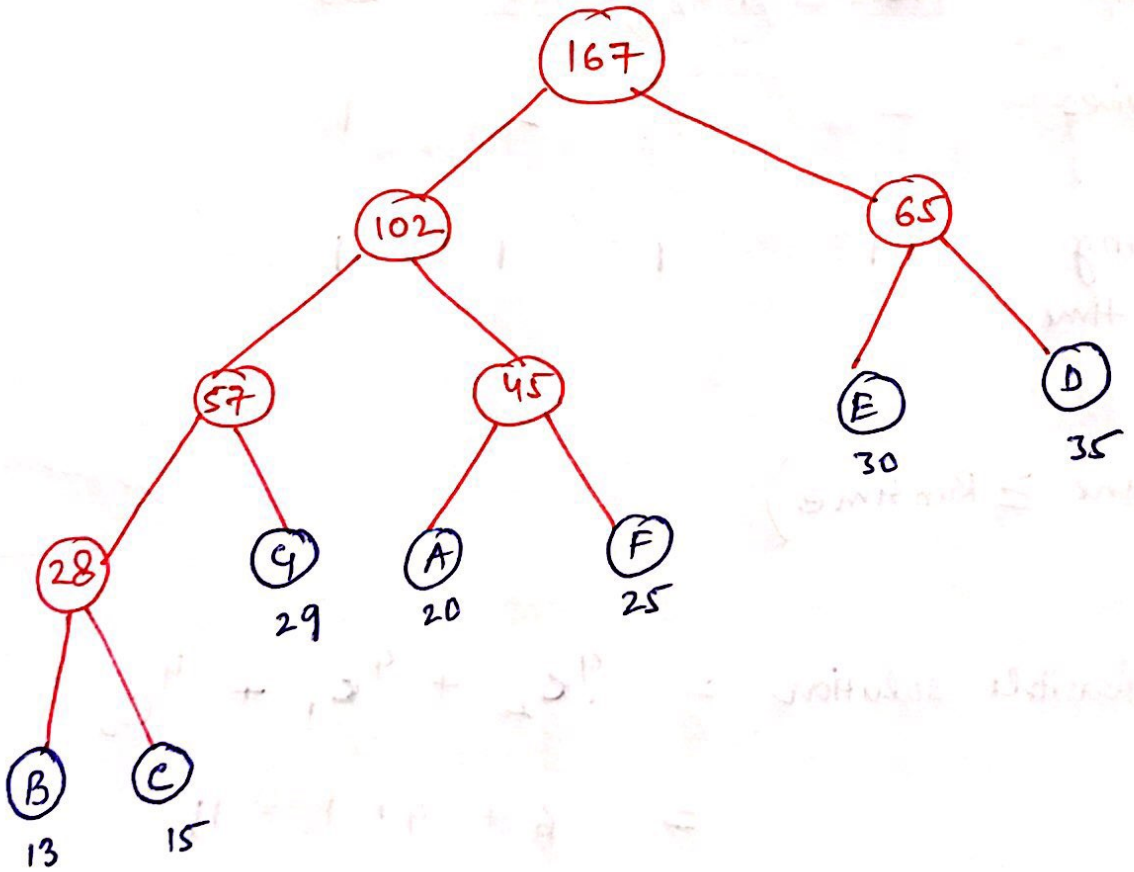
- So, to find optimal merge pattern :

→ starts with merging 2 minimum (same as huffman coding)

eg:-

$n = 7$

A = 20	D = 35
B = 15	E = 30
C = 13	F = 25
G = 29	



Total mov op^n = 28 + 45 + 57 + 65 + 102 + 167

= 464

Job Sequencing with deadlines :-

(i) single CPU

(ii) No - premption

(iii) One unit running time to each job

(iv) Arrival time of each job same

eg:- $n = 4$

Jobs	J_1	J_2	J_3	J_4
Profit	175	125	150	200
Deadlines	2	1	2	1
Running time	1	1	1	1

(Deadline $\geq$ Runtime)

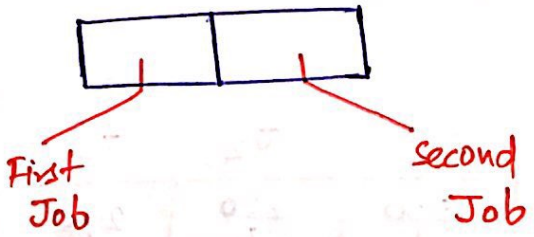
Total feasible solution $= {}^4C_2 + {}^4C_1 + {}^4C_0$

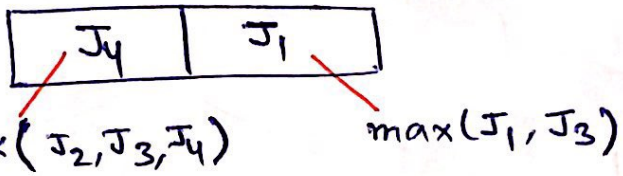
$= 6 + 4 + 1 = 11$

Optimal solution $= (J_4, J_1) = 375$

- Maximum possible job completion = $\frac{\text{Maximum deadline}}{\text{Running time}}$

- Make array for job completion :





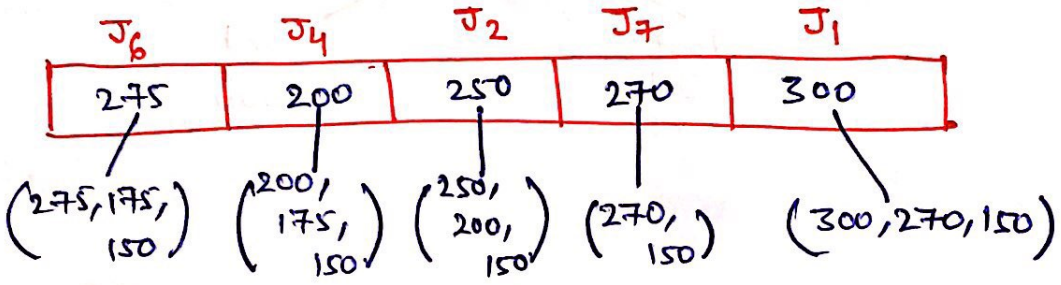
Q. $n = 7$

Jobs :	J_1	J_2	J_3	J_4	J_5	J_6	J_7
Profit :	300	250	150	200	175	275	270
deadline :	5	3	5	3	2	1	5

Maximum job completion = $\frac{5}{1}$ = 5

Jobs with deadline

1,2,3,4,5	2,3,4,5	3,4,5	4,5	≥5



Q. $n = 9$

Jobs :	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
Profit :	75	50	25	70	55	30	60	45	68
Deadlines :	5	3	7	3	5	4	2	5	3

(i) sort in decreasing order of profit:

J_1	J_4	J_9	J_7	J_5	J_2	J_8	J_6	J_3
75	70	68	60	55	50	45	30	25
5	3	3	2	5	3	5	4	7

J_7	J_9	J_4	J_5	J_1		J_3
60	68	70	55	75	X	25

~~50~~ 45 ~~60~~ 30

50 ~~68~~ 45 30 60

50 ~~70~~ 68 45 30

~~55~~ 45 30

~~75~~ 55 45

25

(Maximum profit = 353)

(penality = 125)

Dynamic programming :-

Greedy Technique	Few possibilities	Less time	Sometimes wrong
Dynamic Problem	All possibilities	More time	Always correct

Applications of Dynamic problem :-

(i) Fibinocci series

(ii) Longest common subsequence (L.C.S)

(iii) 0/1 knapsack

(iv) Matrix chain multiplication

(v) Sum of subsets problem

(vi) All pairs shortest path

(vii) Optimal cost BST

(i) Fibinocci Series :-

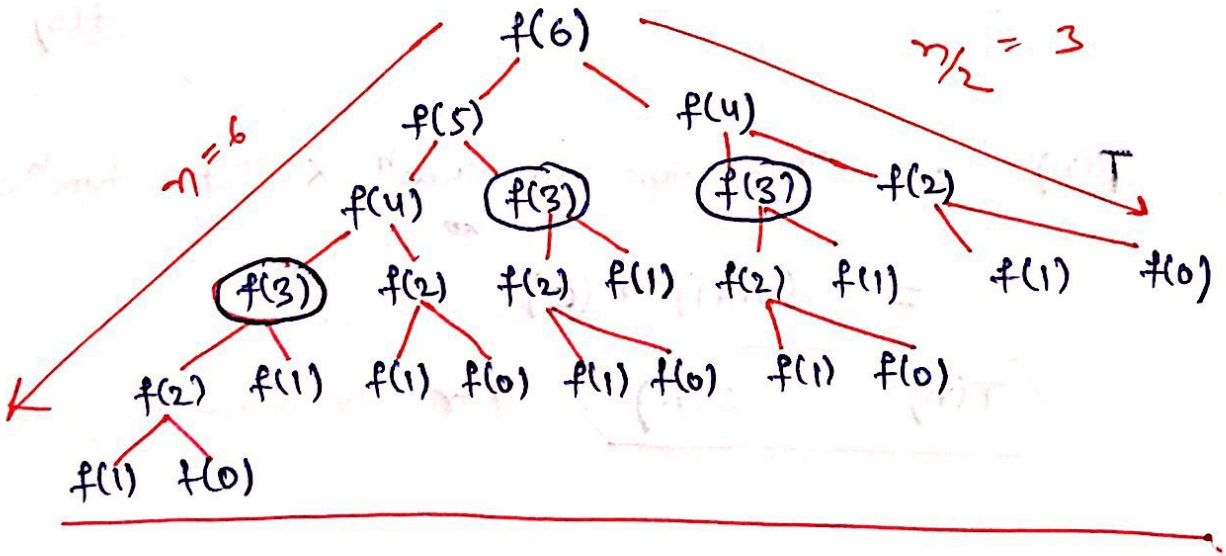
n	0	1	2	3	4	5	6	7	8	9	10
fib(n)	0	1	1	2	3	5	8	13	21	34	55

```
fib(n)
{
    if(n==0)
        return 0;
    else if(n==1)
        return 1;
    else
        return fib(n-1) + fib(n-2);
}
```

Recurrence Relation for time:-

$$T(n) = \begin{cases} O(1) & \text{if } n=0 \text{ or } n=1 \\ T(n-1) + T(n-2) + c & \text{if } n>1 \end{cases}$$

$$T(n) = O(2^n)$$



- In the above recursive tree, many func call's are repeating, known as overlapping sub-problem.

- There is no need to compute same func call again & again.

- So, In Dynamic programming we will compute only distinct funcn call, because as soon as we compute any funcn we store results in some datastructure so that we can reuse it in future.

// Distinct funcn call in fibinocci of n :-

$f(n) \rightarrow f(n-1) \rightarrow \text{-----} \rightarrow f(3) \rightarrow f(2) \rightarrow f(1)$
$\downarrow$
$f(0)$

$T(n)$ = Time for one funcn × Total funcn calls

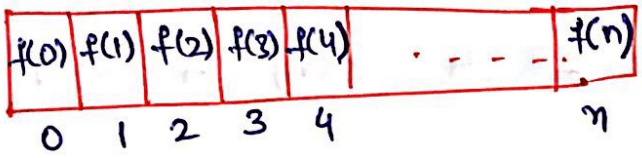
$= (n+1) \cdot (c)$

$T(n) = O(n)$ for dynamic programming

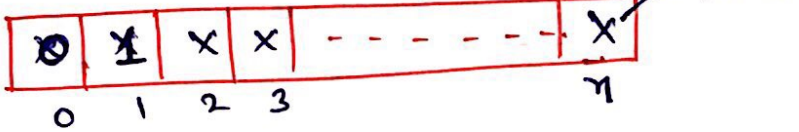
T(n) reduces : [$O(2^n) \longrightarrow O(n)$]

But to store intermediate solution we need an array so that we can take them directly.

Table



Initial



So, In dynamic programming :-

NOTE :- Here conditional recursion occurs.

```
DP_fib(n)
{
  if (n==0 || n==1)
    return n;
  else
    {
      if ( Table[n-1] == NULL)
          Table[n-1] = DP_fib(n-1);   // store in table
      if ( Table[n-2] == NULL)
          Table[n-2] = DP_fib(n-2);   // store in table
      Table[n] = Table[n-1] + Table[n-2];   // store in table
      return Table[n];
    }
}
```

Space complexity :-

$S(n) = I/p + Extra$

- I/p → constant (2-Byte)
- $Extra$ → stack size + Table size

$= 2 + O(n) + O(n)$

$S(n) = O(n)$

14 Jul 2019