Greedy Techniques :-

NOTE:- Most of the problems in greedy contains n inputs and our objective is finding a subset which will satisfy our conditions, and optimizes our goal.

Basics of Greedy Techniques :-

1. Solution space
2. Feasible solution
3. Optimal solution

- Set of all possible solution over the given I/p is known as **Solution space**.
- Set of all solution which will satisfy our conditions known as **Feasible solution**.
- Those feasible solution which optimizes our goal known as **Optimal solution**.

Application :-

(i) knapsack

(ii) Job sequencing with deadlines

(iii) Huffman coding

(iv) Optimal merge pattern

(v) Minimum cost spanning tree

↳ Prism

↳ Kruskals

(vi) single source shortest path

↳ Dijkstra's

↳ Bellmen's ford

↳ Breadth First Traversal (BFS)

(i) Knapsack :-

	OBJ_1	OBJ_2	OBJ_3	OBJ_4
weight	8	3	12	5
cost	200	50	220	190
cost/weight	25	16.66	18.33	38
order	2	4	3	1

Descending order

Let M is the capacity of knapsack -

M = 24 (let)

order : $O_4 \rightarrow O_1 \rightarrow O_3 \rightarrow O_2$

O_4 :

$M = 24 - 5(1)$
$= 19$

Profit = $0 + 190$
$= 190$

O_1 :

$M = 19 - 8(1)$
$= 11$

Profit = $190 + 200$
$= 390$

O_3 :

$M = 11 - (11/12)12$
$M = 0$

fraction

Profit = $390 + (11/12 \times 220)$
$= 591.6$

O_2 :

$M = 0$

$$\begin{pmatrix} 1 & 11/12 & 0 & 1 \\ O_4 & O_3 & O_2 & O_1 \end{pmatrix}$$

eg:2

$n = 7$ (no. of objects) $M = 33$

Object :	O_1	O_2	O_3	O_4	O_5	O_6	O_7
weight :	15	3	9	6	7	8	3
Cost :	75	15	95	65	85	70	60
C/W ratio :	15	5	10.5	10.8	12.14	8.75	20
order :	12	7	5	4	3	6	1

$O(n)$

$O(n \log n)$

O_7 : $m = 33 - 3(1)$ $= 30$ Profit $= 0 + 60$ $= 60$

O_1 : $m = 30 - 5(1)$ $= 25$ Profit $= 60 + 75$ $= 135$

O_5 : $m = 25 - 7(1)$ $= 18$ Profit $= 135 + 85$ $= 220$

O_4 : $m = 18 - 6(1)$ $= 12$ Profit $= 220 + 65$ $= 285$

O_3 : $m = 12 - 9(1)$

$= 3$

Profit = $285 + 95$

$= 380$

O_6 : $m = 3 - 8(3/8)$

$m = 0$

Profit = $380 + (3/8 \times 70)$

$= 406.25$

O_2 : $m = 0$

1	0	1	1	1	3/8	1
O_1	O_2	O_3	O_4	O_5	O_6	O_7

Since For p in knapsack algorithm :

Calculate profit % $\longrightarrow O(n)$

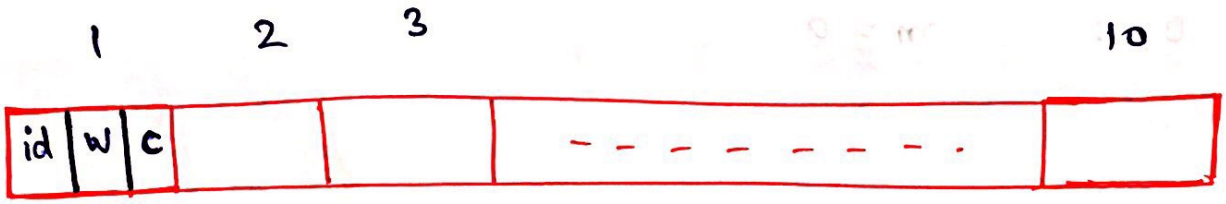
sorting (Descending order) $\longrightarrow O(n \log n)$

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

In knapsack problem, there is only array of structures:

```
struct obj
{   char id;
    int weight;
    int cost;
};
```

obj A[10];



(ii) Single Source shortest path:-

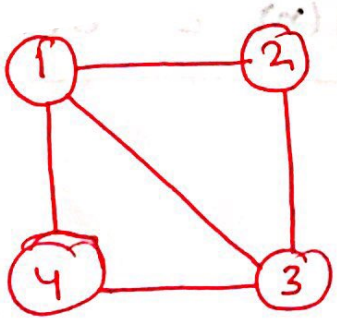
Graph Introduction:-

G(V, E)

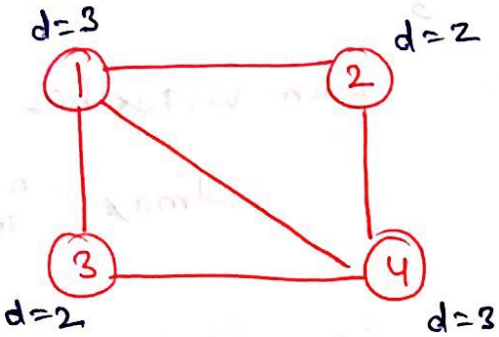
- E: Set of Edges
- V: Set of Vertex

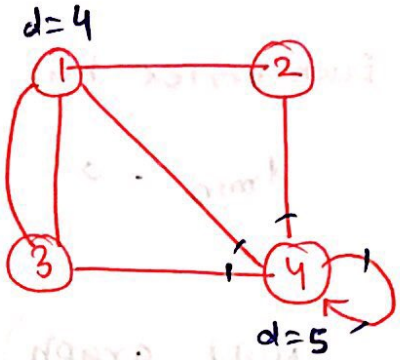
V = {1, 2, 3, 4}

E = {e1, e2, e3, e4, e5}



Simple Graph	Multi Graph
✓ No self Loop	✓ Self loop
✓ No parallel edges	✓ parallel edges





* A simple graph with n vertices maximum degree is $n-1$ and minimum is 0

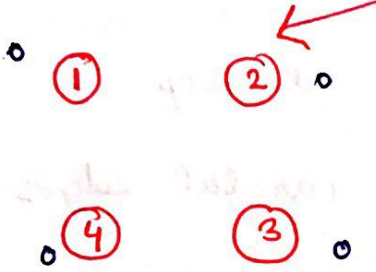
$$\begin{cases} d_{max} = n-1 \\ d_{min} = 0 \end{cases}$$

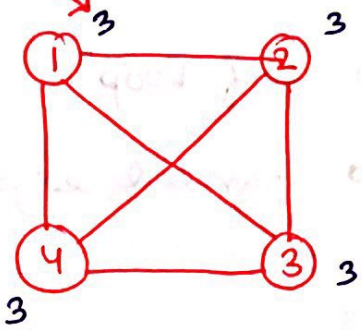
if $d_{max} > n-1$ then graph is a multigraph

* For a multigraph:

$$\begin{cases} d_{max} = \infty \\ d_{min} = 0 \end{cases}$$

Simple graph





Each vertex has

$d_{min} = 0$

(NULL graph)

Each vertex has

$d_{max} = n-1$
$= 4-1 = 3$

(COMPLETE graph)

$$\frac{\text{Sum of degree}}{2} = \text{No. of edges}$$

Let $G(V,E)$ be a simple Graph then,

$$0 \leq E \leq \frac{V(V-1)}{2}$$

so, $E = cV^2$

$E = O(V^2)$

$\therefore \log E = O(\log V)$

$E = cV^2$

$\log E = 2c \log V$

$2c$ → const

No. of simple graph possible with n-vertices :-

$$m = \text{max edges} = \frac{n(n-1)}{2}$$

$$\Rightarrow \left({}^{m}C_0 + {}^{m}C_1 + {}^{m}C_2 + \cdots\cdots {}^{m}C_{n-1} + {}^{m}C_n \right)$$

for $n = 5$

$$m = \frac{5 \times 4}{2} = 10$$

OR

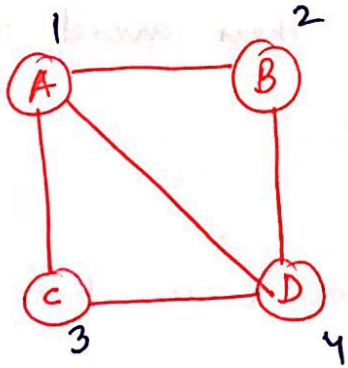
$$\left(\text{possible simple graph} = 2^m \right)$$

Graph representation :-

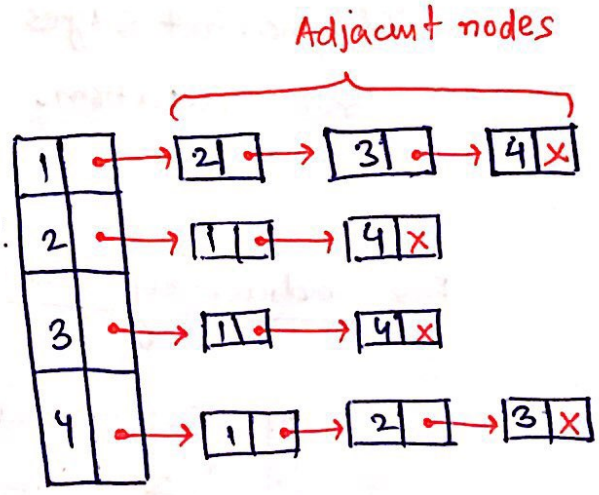
(i) Adjacency matrix (static)

(ii) Adjacency list (Dynamic)

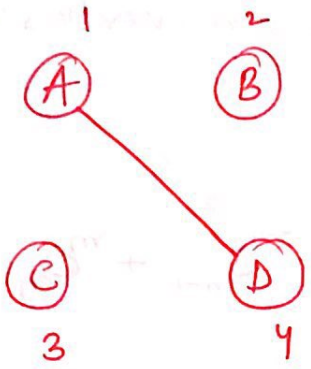
eg:-



$$Q = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$



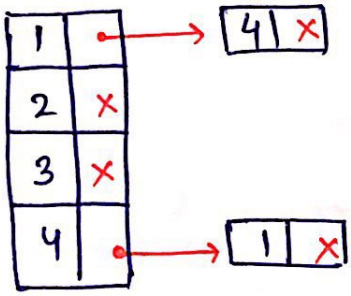
eg:



Adjacency matrix

$$= \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

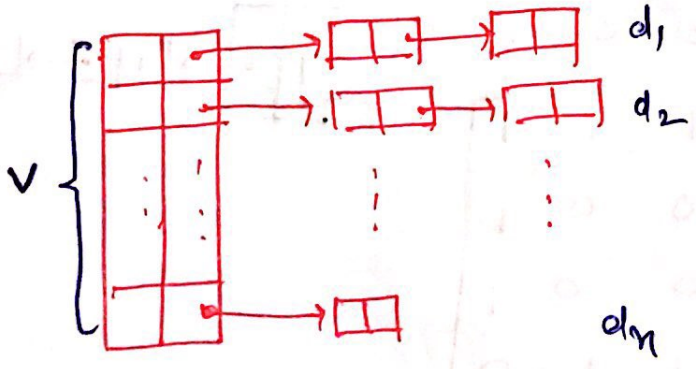
Adjacency List :-



NOTE :- Adjacency matrix is best for storing dense graph as it always takes space $O(n^2)$.

If no. of edges is less then avoid this representation.

For adjacency list, space required is = $O(V+E)$



$d_1 + d_2 + \cdots d_n = 2E$

So, space = $V + 2E$

$= \boxed{O(V+E)}$

Sparse graph
(less edges)

↑

(Adjacency List)

Dense graph + complete graph
(more edges)

↑

(Adjacency matrix)

To check adjacent nodes (A–B) :-

- Adjacency matrix
  - O(1) ← All case
- Adjacency List
  - O(n) ← Worst case
  - O(1) ← Best case

To check degree of nodes :-

	Worst case	Best case
Adjacency matrix	O(n)	O(n)
Adjacency List	O(n)	O(1)

To calculate no. of edges :-

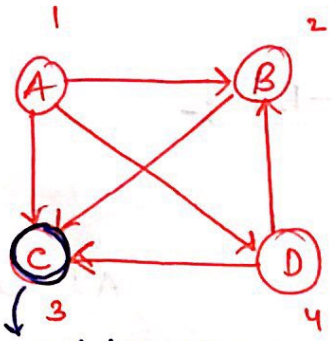
Adjacency matrix → $O(V^2)$

Adjacency list → $O(V+E)$

- In directed graph -

(Sum of Outdegree / Indegree = E)

- Universal sink -



outdegree = 0

Indegree = n-1

Time complexity to find a universal sink in a graph = $O(V^2)$ (matrix)

$$G = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Indegree (d^-)

out degree (d^+)

$A \to B \longrightarrow G[1,2] = 1 \ (d_A^+ > 0) \longrightarrow$ A not possible $\to O(1)$

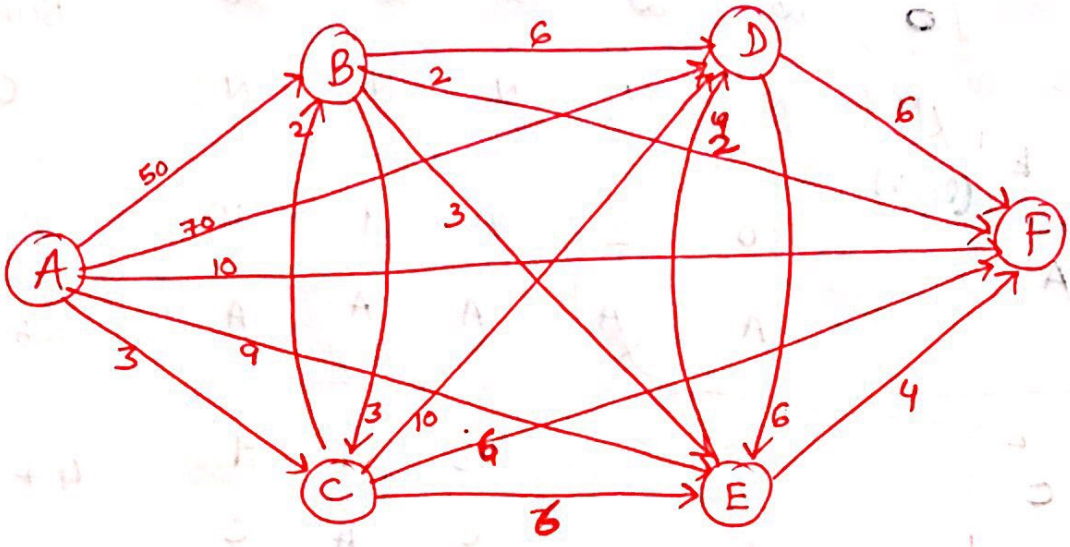
$B \to C \longrightarrow G[2,3] = 1 \ (d_B^+ > 0) \longrightarrow$ B not possible $\to O(1)$

$C \to D \longrightarrow G[3,4] = 0 \longrightarrow$ check column (C) if sum is $(V-1)$ then C is universal sink

$T(n) = V + O(1) + O(1)$

$T(n) = O(V)$

Single source shortest path :-



$A \to A = 0$

$A \to B = 5 \quad (A \to C \to B)$

$A \to C = 3$

$A \to D = 10 \quad (A \to C \to B \to E \to D)$

$A \to E = 8 \quad (A \to C \to B \to E)$

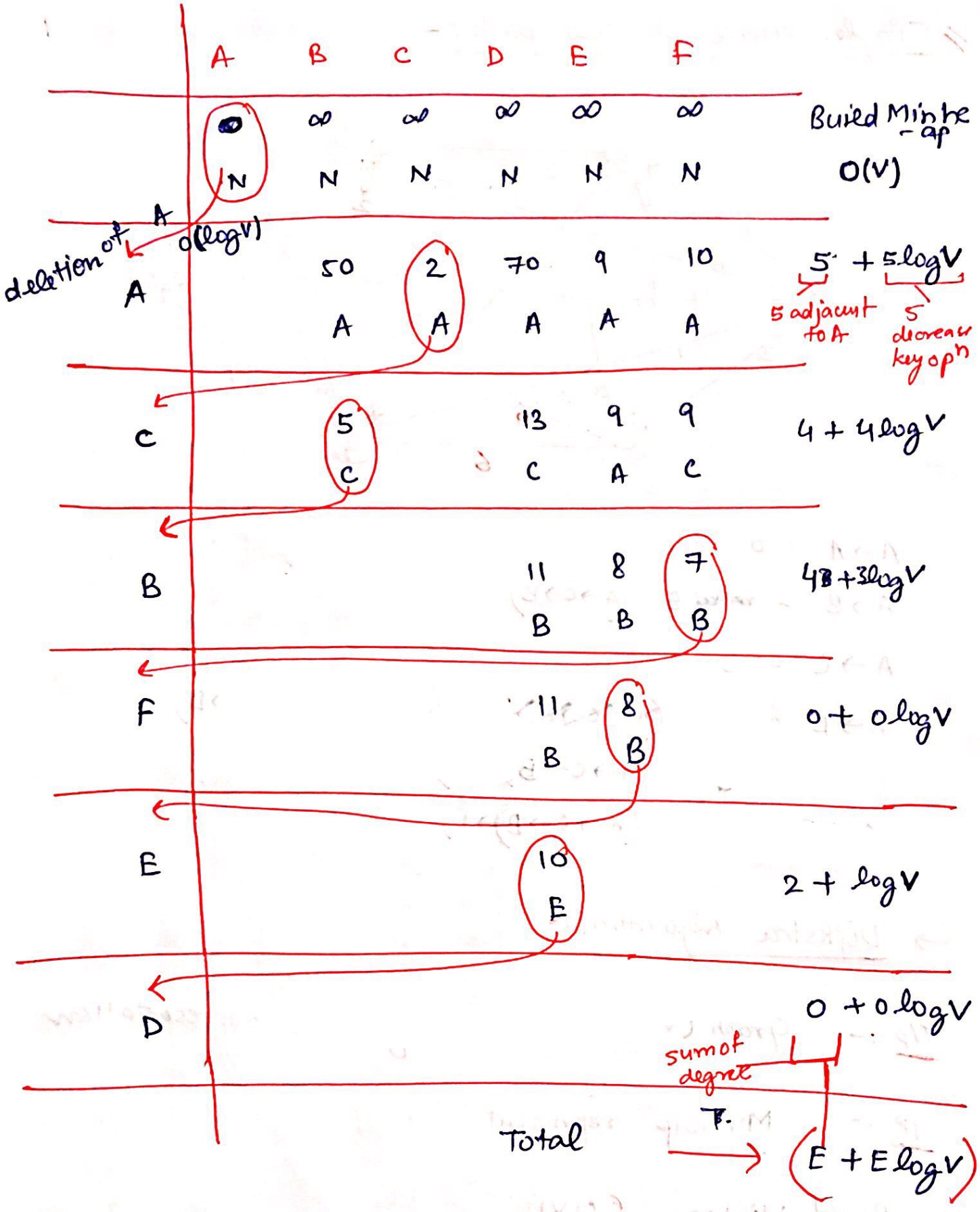
$A \to F = 7 \quad (A \to C \to B \to F)$

↳ Dijkstra Algorithm :-

I/p :- Graph (V, E) in adjacency list representation

O/p :- Minheap representation

- Build Minheap $(O(V))$
- Perform deletion $(O(\log V))$
- Take adjacent vertex and perform decrease key of minheap. on each of them $(O(E \log V))$



O/P

A	B	C	D	E	F	
0	5	2	10	8	7	distance Min from source
X	C	A	E	B	B	Parent

Time complexity = $O(V) + E + E\log V + V\log V$

- $O(V)$: Build Heap
- E : calculate adjacent
- $E\log V$: decrease key for each adjacent vertex
- $V\log V$: Perform deletion on minheap for each vertex

$= V\log V + E\log V$

$= O((V+E)\log V)$

// Path :- F → B → C → A

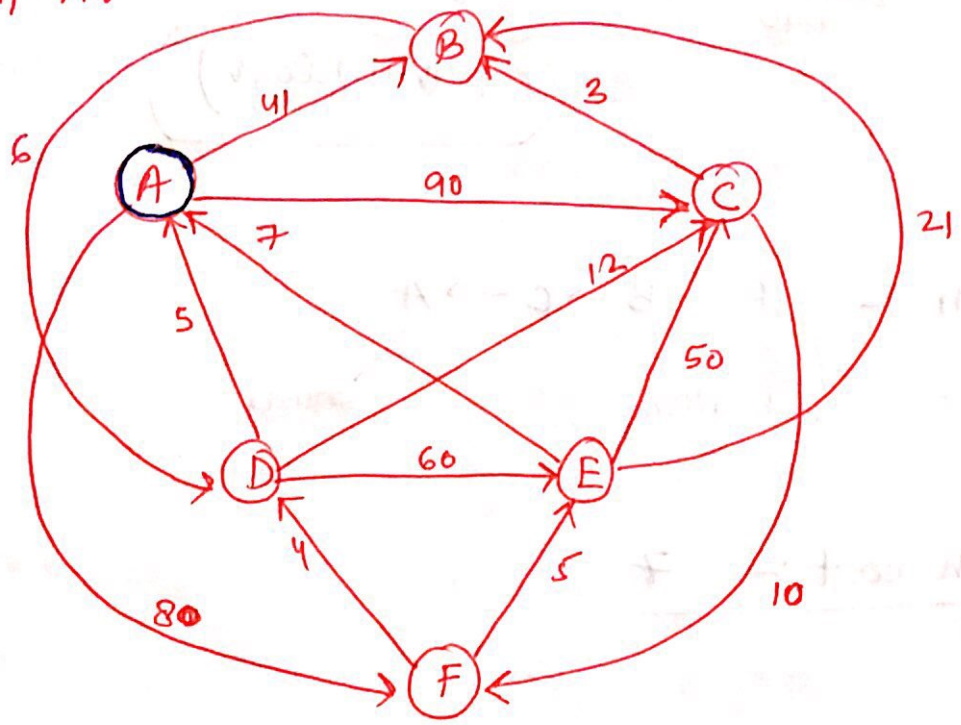
F : destination, A : source

// Path cost :- 7

So, Time complexity of Dijkstra algo :-

If I/p is adjacency List	If I/p is adjacency matrix
$T(n) = O(V) + E + E\log V + V\log V$	$T(n) = O(V) + V^2 + E\log V + V\log V$
E : calculate adjacent total	V^2 : calculate total adjacent
$T(n) = O((V+E)\log V)$	$T(n) = O(V^2 + E\log V)$

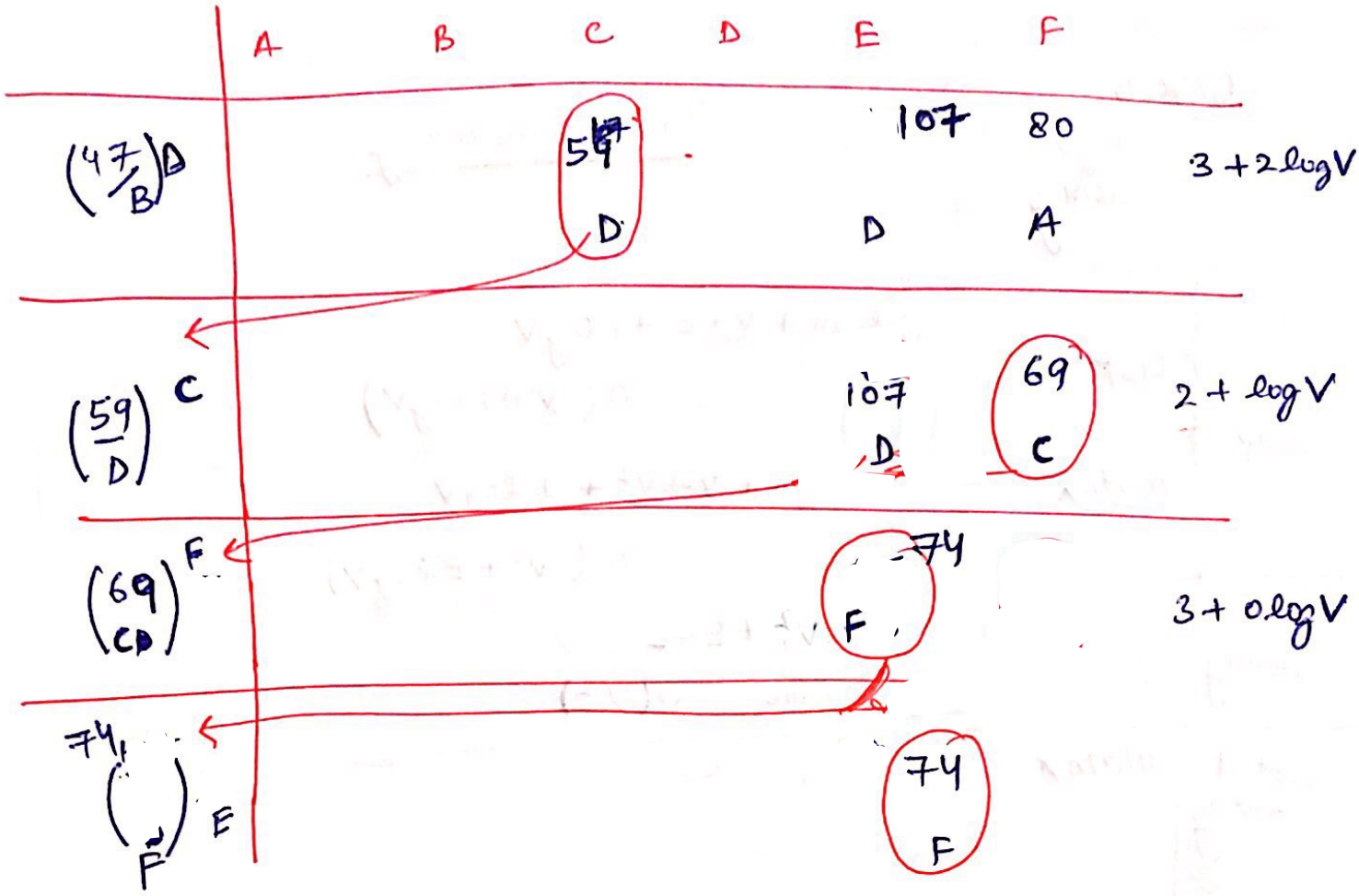
Q.1 Apply Dijkstra algo to ~~find calculate minimum cost path~~ (a) Print the sequence of vertexics when started from A.



(b) what will be the cost of shortest path from A to E

(c) what is shortest path from A to E

	A	B	C	D	E	F	
	0	∞	∞	∞	∞	∞	O(V)
	N	N	N	N	N	N	
(0, N) A		41	90	∞	∞	80	3 + 3log V
		A	A	N	N	A	
(41, A) B			90	~~45~~	∞	80	1 + log V
			A	B	N	A	



Sequence of vertices :-

A	B	D	C	F	E
0	41	47	59	69	74
x	A	B	D	C	F

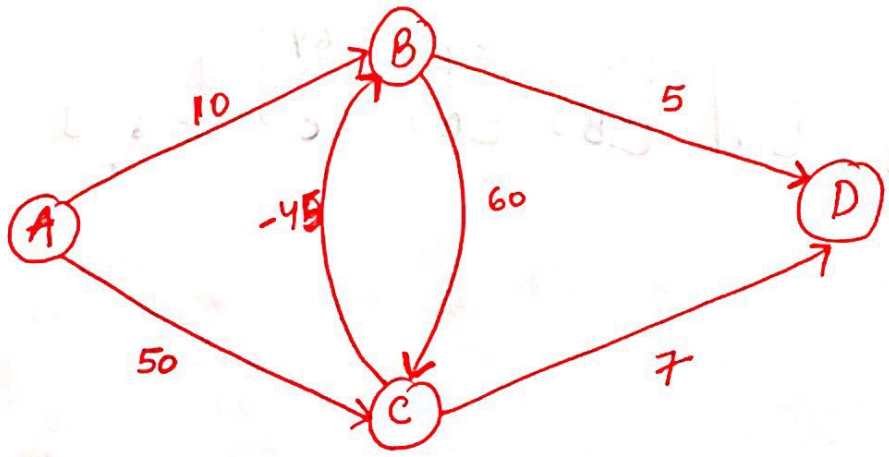
(b) 74

(c) E ← F ← C ← D ← B ← A

NOTE :- Dijkstra algo only applicable for positive weighted Graph.

Dijkstra algo using		Time-complexity
Min Heap	List	$V\log n + V + E + E\log V = O((V+E)\log V)$
	Matrix	$V\log n + V + V^2 + E\log V = O(V^2 + E\log V)$
array	List	$V^2 + E + E + V$ (V^2: selection sort pass) $= O(V^2)$
sorted array	Matrix	$V + V^2 + EV + V = O(V^2 + EV)$

Q.2.



Apply Dijstra :-

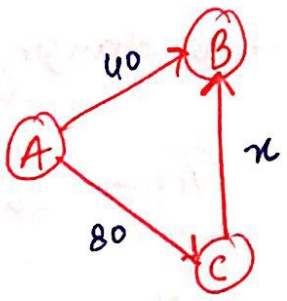
NOTE:- Relaxation at A is done when all the vertices adjacent to A perform decrease key

In Dijkstra every node will be relaxed once.

	A	B	C	D
	0	∞	∞	∞
	N	N	N	N
(0, N) A		10	50	∞
		A	A	N
(10, A) B			70	15
			B	B
(15, B) D			70	
			B	
(70, B) C				

As, we can see cost for vertex B is 10 due to dijkstra which is wrong as actual cost (minimum) is 5.

This is because vertex B is eliminated as dijstra expected that some +ve weight will came in further processing.



$80 + x < 40$

if $(x < -40)$ i.e -ve

if $x = -50$

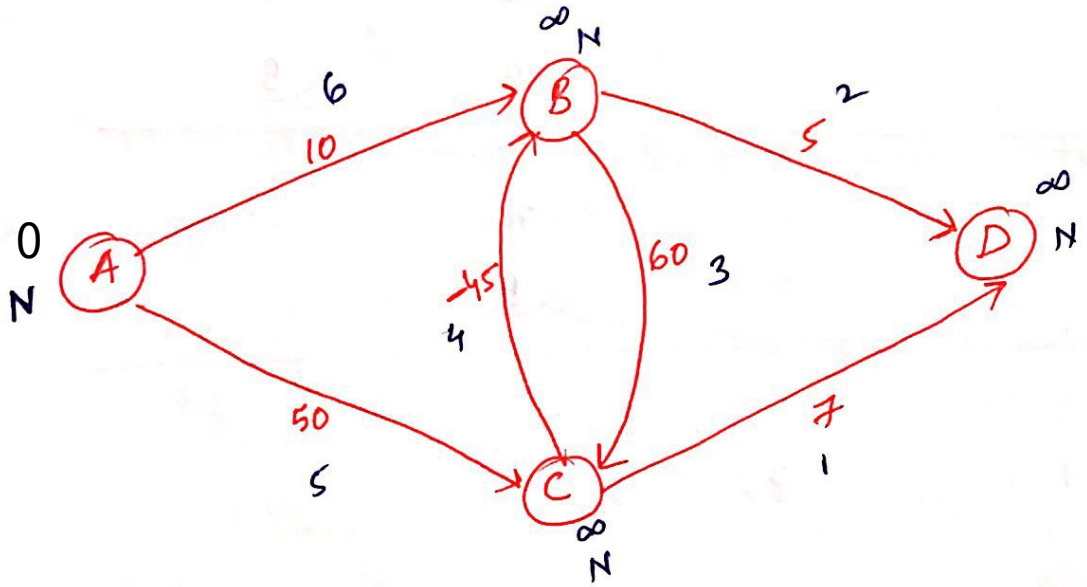
then minimum cost for B

$= 30$

- To eliminate this problem we use bellman's ford algorithm.

⇒ Bellmen's ford algo:-

In dijstra algo each vertex is not optimized once it is deleted from min heap, but in this algo can further optimize if -ve weight is there.

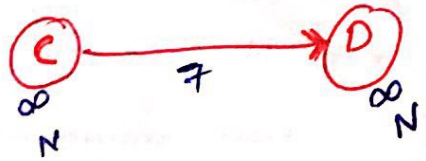


Steps:-

(i) Give numbers to each edge (numbering of edges).

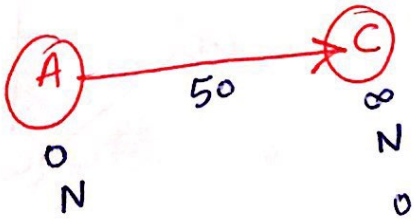
(ii) Perform decrease key oph for each edges in numbering order.

(iii) Repeat (ii) until vertex state changing i.e (V-1).



∞+7 = ∞

N



0+50 = (50, A)

A

round	A	B	C	D	
Initial	0 N	∞ N	∞ N	∞ N	normal array
I 1	0 N	10 A	50 A	∞ N	E · O(1) decrease key op^n in array
2	0 N	5 C	50 A	15 B	E · O(1)
3	0 N	5 C	50 A	10 B	E · O(1)
4	0 N	5 C	50 A	10 B	same as 3rd round stop

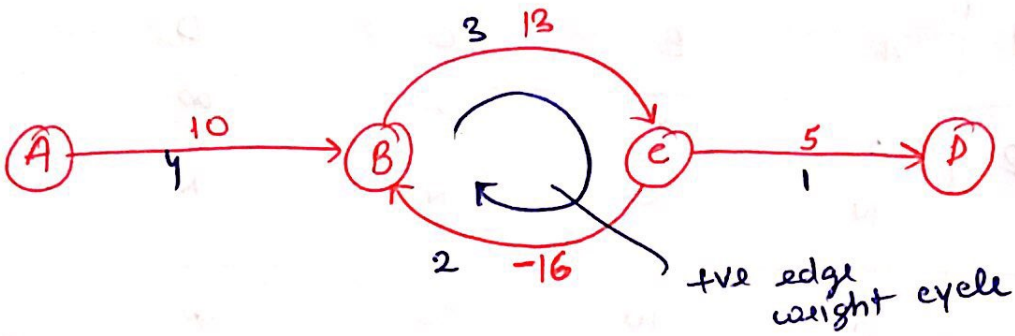
V-1 rounds

Time-complexity = (V-1) O(E)

$T(n) = O(VE)$

NOTE:- If graph contains all +ve weight, then dijkstra is more efficient but in case of -ve weight dijkstra failed so apply Bellman's ford.

Q.



(a) Apply Dijkstra:

	A	B	C	D
	0 (N)	∞ N	∞ N	∞ N
(0, N) A		10 A	∞ N	∞ N
(10, A) B			23 B	∞ N
(23, B) C				28 C
(28, C) D				

(b) Apply Bellmen's ford :-

round	A	B	C	D	
Initial	0	∞	∞	∞	
	N	N	N	N	
1	0	10	∞	∞	O(E)
	N	A	N	N	
2	0	10	23	∞	O(E)
	N	A	B	N	
3	0	7	20	28	O(E)
	N	C	B	C	
Verification round 4	0	4	17	25	O(E)
	N	C	B	C	
∞	0	$-\infty$	$-\infty$	$-\infty$	O(E)
	N	C	B	C	

V-1 rounds

∞ times

NOTE:- Due to presence of -ve weight cycle, vertex associated with -ve weight cycle B, C (directly) & D (indirectly) will continue decrease key op^n until $-\infty$.

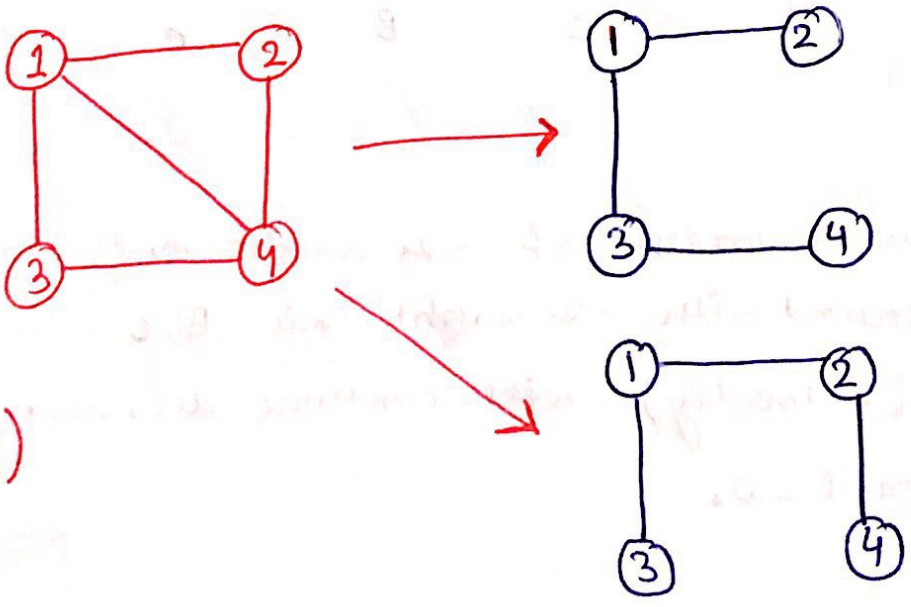
- Thus in case of -ve edge weight cycle bellmen's ford algorithm fails as it goes into a infinite cycle.

- Bellmen's ford algo will give right answer only for those vertex which doesn't belong to -ve edge weight cycle.

- Bellmen's ford algo can detict all -ve weight edge cycle if they are reachable from source.

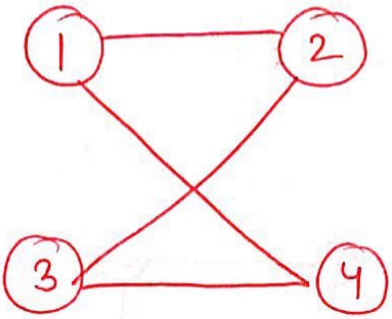
(iii) Minimum Cost spanning tree :-

A subgraph S of the given graph G(V,E) is said to be spanning tree ; S should contain all vertices of G with using minimum no. of edges without cycle, is known as spanning tree.



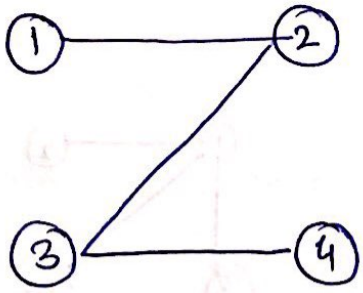
(E = V-1)

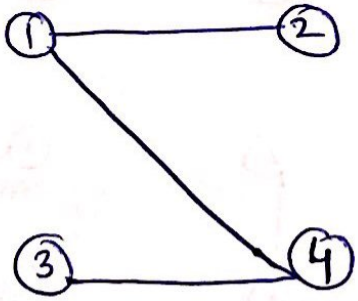
eg:- Find number of spanning tree for the given graph?

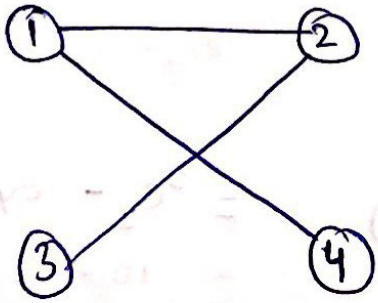


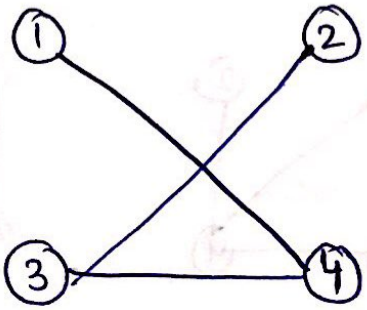
$^4C_3 = 4$

- Cycles

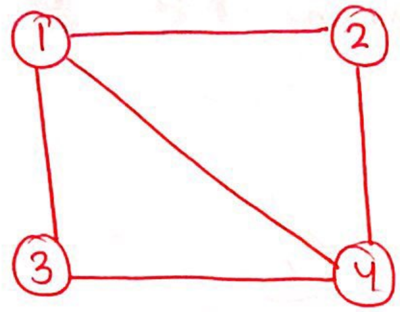


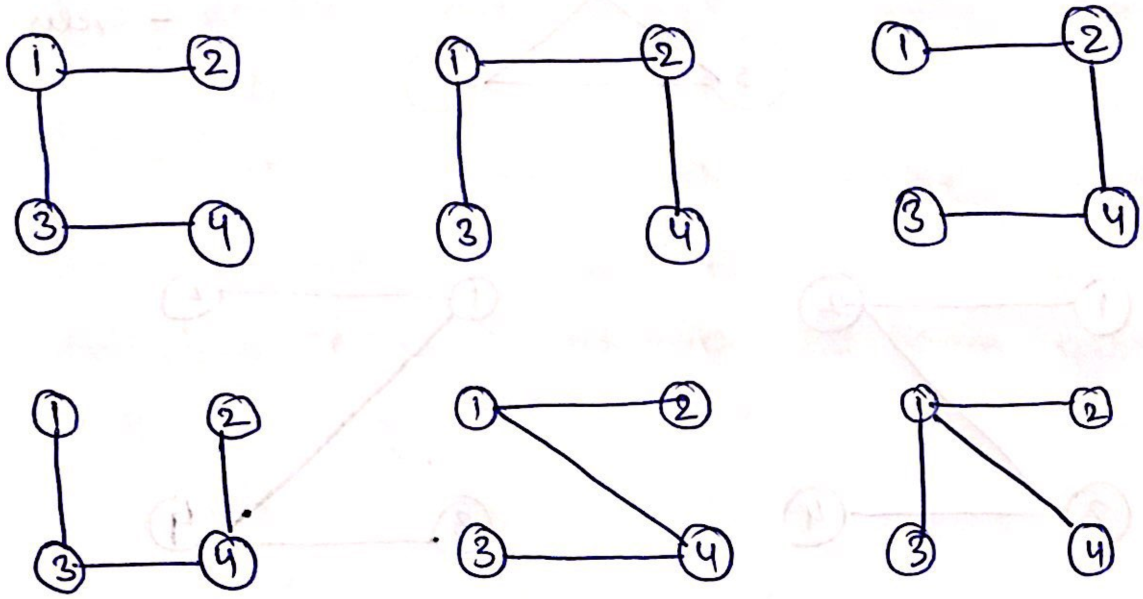


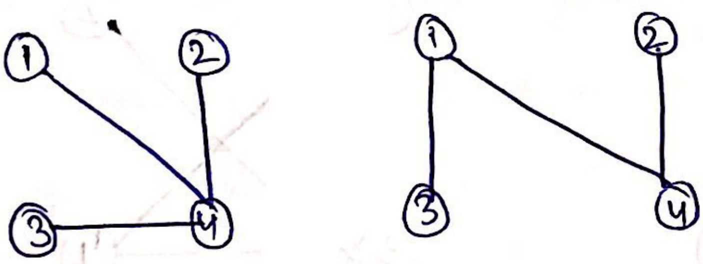




eg:- Find number of spanning tree:



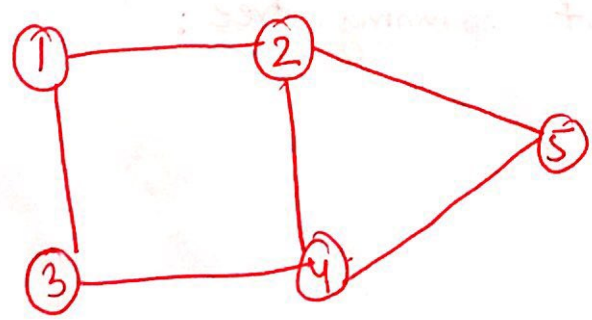




$= {}^5C_3 -$ cycles

$= 10 - 2 = \underline{8}$

eg:-



$= {}^{6}C_{4} - (\text{cycles} + \text{disconnected graph})$

$= \frac{6 \times 5}{2} - (2 + 2)$

$= 15 - 4 = 11$

* No. of spanning trees for complete graph of n vertices $= (n^{n-2})$

= To construct minimum cost spanning tree for given graph, we have two algorithms:

↳ Kruskal's

↳ Prim's

↳ **Kruskal's algorithm :-**

(i) Make individual sets of each vertices.

(ii) Generate Min heap for the edges.

(iii) Perform deletion on Minheap and perform union of corresponding vertices sets.

(iv) Before adding edge to spanning tree

call FindSet() func to check that vertics are in different group.

- if, vertics are in different group then add edges to spanning tree and call makeset() func.
- otherwise perform next deletion from minheap

(V) Repeat until all vertics comes under same set or group.

Time complexity

Best case

- $E + (V-1)\log E$

Build Heap — deletion minheap

$(O(E + V\log E))$

* V-1 times deletion

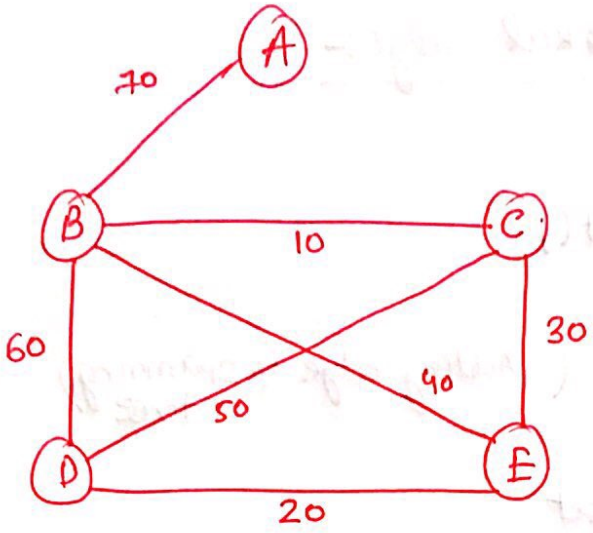
Worst case

- $E + E\log E$

$(O(E\log E))$

* E times deletion

eg:-



A

B — 10 — C

D E

A D E (B, C)

(i)

A

B — 10 — C

D — 20 — E

A (B,C)
(D,E)

(ii)

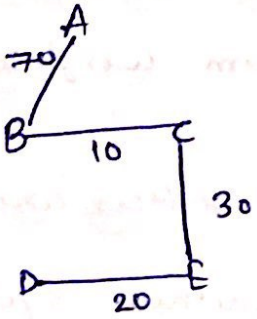
A

B — 10 — C
30
D — 20 — E

(B,C,D,E)
A

(iii)

(iv)



(A, B, C, D, E)

stop

(cost = 130)

Basic op^n in Kruskal algo:-

↳ FindSet()

↳ Union (Adding edge to spanning tree) — O(1) using union by Rank & path compression

↳ Makeset

NOTE:- In Kruskal method we have disconnected graph in intermediate of op^n but in prims method we have always connected graph throughout op^n.

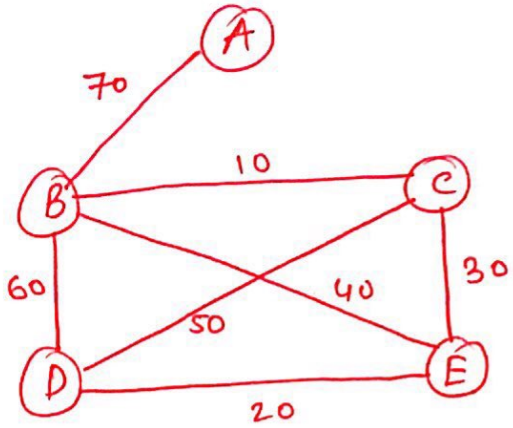
↳ Prims algorithm:-

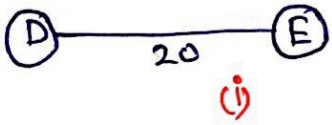
(i) Take any vertex from the graph

(ii) Find all the adjacent vertices for the choosen vertex and select minimum weighted edge

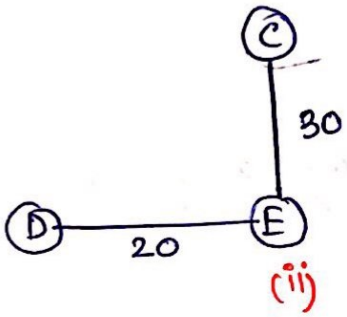
(iii) Repeat (ii) until all vertices are covered.

Before selecting edge check whether cycle should not form.

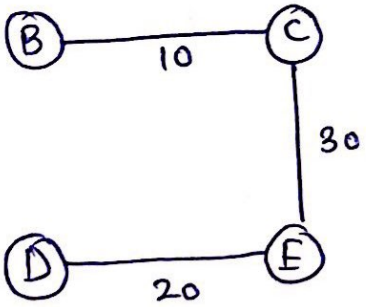




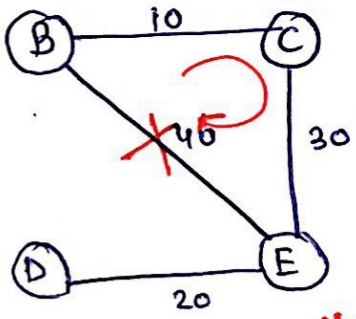
(i)



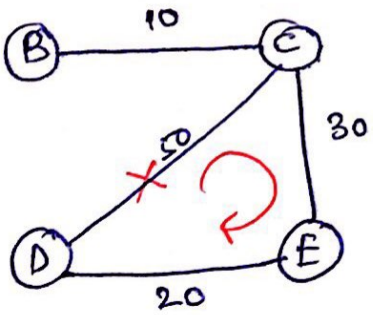
(ii)



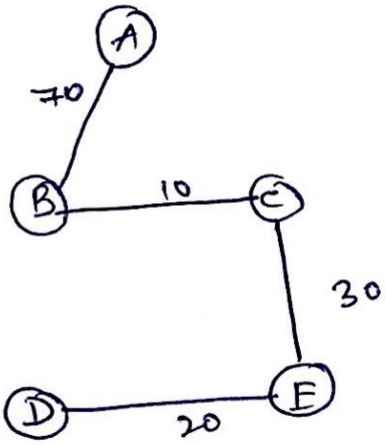
(iii)



(iv)



(v)



(vi)

13 Jul 2019