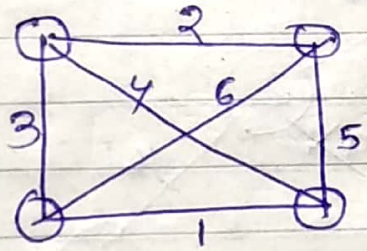
7/8/18

* Graph can have cycle but tree cannot have cycle.

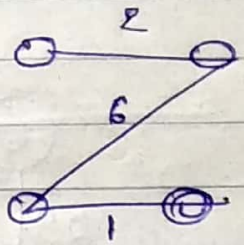
Finding a minimum spanning tree in a graph:-

Assume there is an undirected connected graph G. A spanning tree is a subgraph of G which is a tree and contains all the vertices of G.

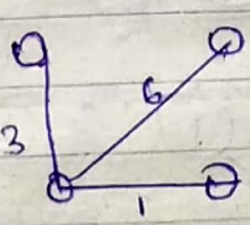
A minimum spanning tree is a spanning tree which has some wts or lengths associated with edges such that total weight of tree is minimum.



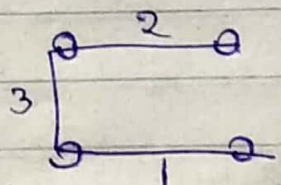
graph G



① Minimum spanning tree of that graph



3rd spanning tree.



Another minimum spanning tree of that graph.

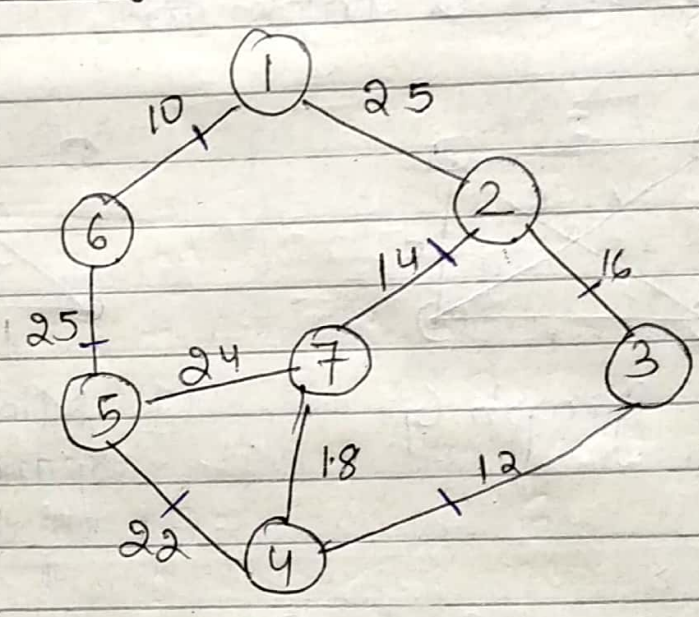
minimum spanning tree.

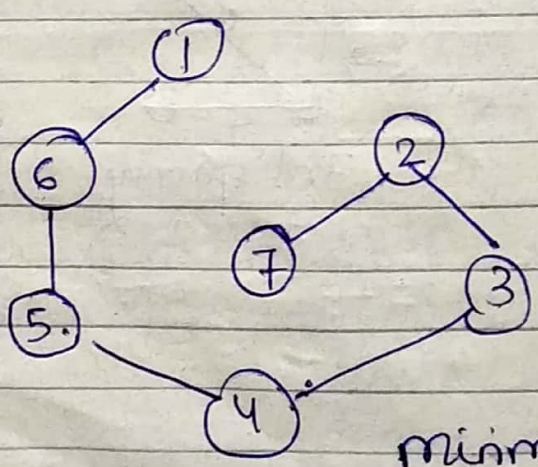
1st greedy approach

→ Pick one edge after another in acceding order till a cycle form. This is Kruskal's algo

2nd ⇒ Another greedy approach is prim's algo

Prim's Algo





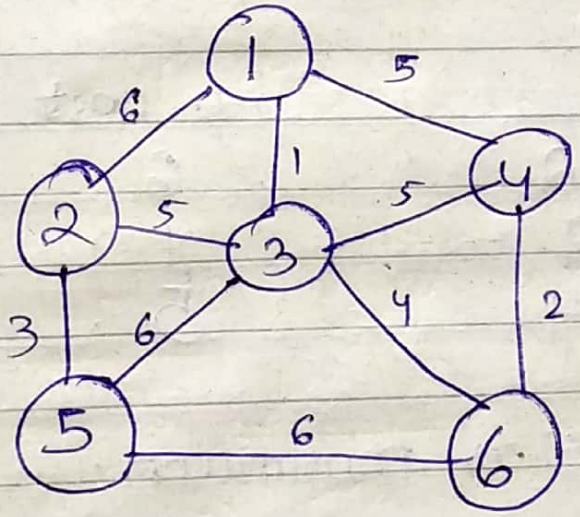
minimum spanning tree

Algorithm:

(Beauty of this algo → you don't have to track cycle, it will track on its own but its complexity is little high).

Prim's

```
T = ϕ
U = {1}          (// Starting node for the graph).
while (U ≠ V)
{
    let (u,v) be the low cost edge
    such that u ∈ U & v ∈ V-U
        T = T U {(u,v)}
        U = U U {v}
}                    // V - set of vertices.
```



To implement this we require 2 arrays:-

① closest [i] or neighbour.
For any i belonging to V-U,
closest [i] gives the vertex in U

that is closest to node i.

Eg:
closest [3] = {1, 2, 4, 5, 6}.

(ii) low cost [i], for any $i \in V-U$, low cost [i] gives the cost of edge between i and closest [i]
edge (i, closest [i])

lowcost [3] = [1, 5, 5, 6, 4]

Implementing

(1) $U = \{1\}$
$V - U = \{2, 3, 4, 5, 6\}$

	Closest	low cost
(u,v)	1 2	6
(u,v)	1 3	1
(u,v)	1 4	5.

So (1, 3) is minimum.

$T = \{(1,3)\}$

(2) $U = \{1, 3\}$.
$V - U = \{2, 4, 5, 6\}$

Closet	Low cost
1 2	6
1 4	5
3 2	5
3 4	5
3 5	6
3 6	4

minimum.

③ $T = \{(1,3); (3,6)\}$

$U = \{1, 3, 6\}$

$V - U = \{2, 4, 5\}$.

closet	low cost
1 2	6
1 3	5
3 2	5
3 4	5
3 5	6
6 4	2
6 5	6

low cost.

④ $T = \{(1,3); (6,4); (3,1)\}$

$U = \{1, 3, 4, 6\}$

$V - U = \{2, 5\}$.

Closest	lowcost
1	6
3	5
2	
3	6
5	6
6	

(3) T = { same + (1,2) }

U = {1,3,6,4,2}

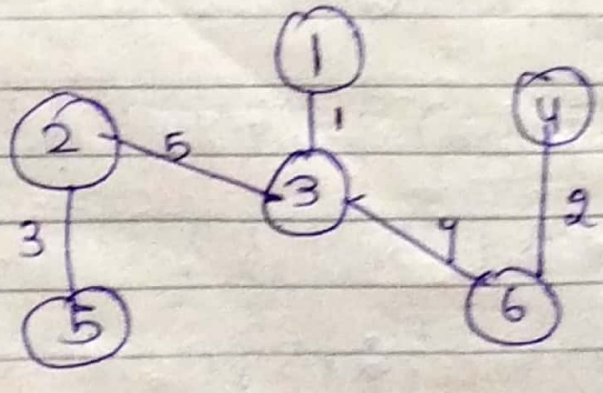
V = {5}.

Closet	lowcost
3	6
5	6
6	
2	3
5	

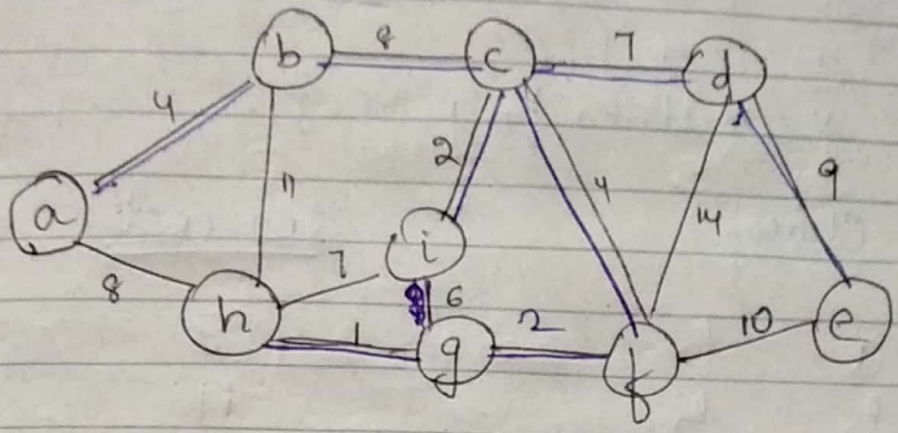
T = { same + (2,5) }.

V = ϕ.

So now minimum spanning tree is :-



2.



① $T = \phi$

$U = \{a\}$

$V = \{b, c, d, e, f, g, h, i\}$

closest		low cost
a	b	4
a	h	8

minimum = 4

② $T = \{(a, b)\}$

$U = \{a, b\}$

$V = \{c, d, e, f, g, h, i\}$.

closest		low cost
a	h	8
b	c	8
b	h	11

(b, c) minimum.

③ T = {(a,b); (b,c)}.

U = {a, b, c}

V = {d, e, f, g, h, i}.

Closest		low cost
a	h	8
b	h	11
c	d	7
c	i	2

③ T = {(a,b); (b,c); (c,i)}

U = {a, b, c, i}

V = {d, e, f, g, h}.

Closest		low cost
a	h	8
b	h	11
c	d	7
i	h	7.
i	g	6
c	f	4

④ $T = \{(a,b); (b,c); (c,i); (i,g)\}$.

$U = \{a, b, c, i, g\}$.

$V = \{d, e, g, h\}$.

closest		low cost
a	h	8
b	h	11
c	d	7
i	h	7
g	h	10
	d	2
		14
f	d	6

⑤ $T = \{(a,b); (b,c); (c,i); (i,g); (g,f)\}$

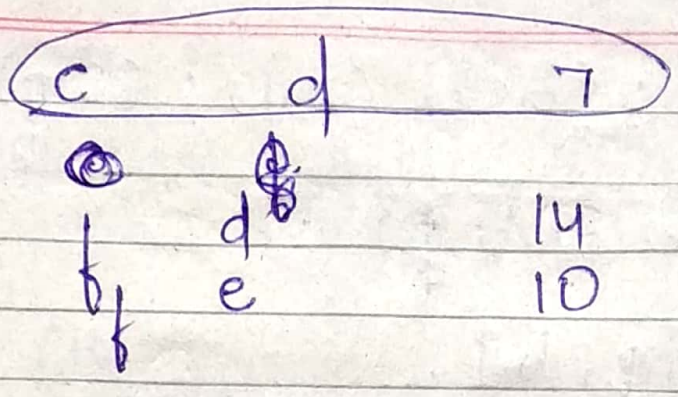
$U = \{a, b, c, i, g, f\}$.

$V = \{d, e, h\}$.

closest		low cost
c	d	7
g	h	2
f	d	
f	e	

⑥ $T = \{$ same $+ (g,h)\}$.

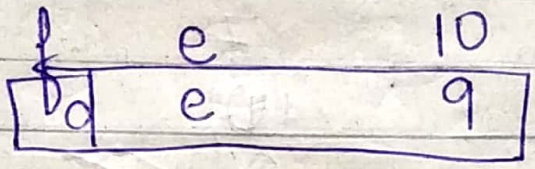
$U = \{a, b, c, i, g, h, f\}$; $V = \{d, e\}$.



(7) - $T = \{sam + (c,d)\}$.

$U = \{a, b, c, d, i, g, h, f, d\}$

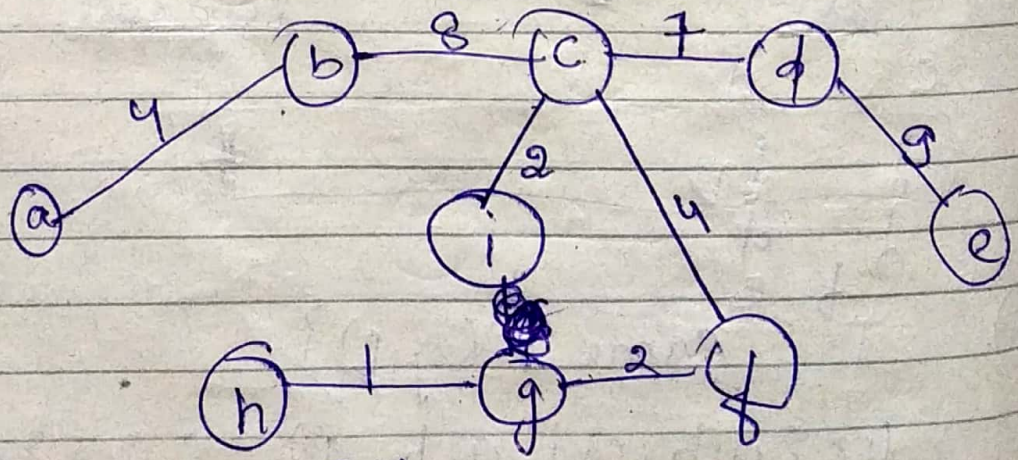
$V = \{e\}$.



(8) $U = \{a, b, c, i, g, h, f, d, e\}$

$V = \{\phi\}$.

Our algo stops here.



Total wts = 37 → Ans

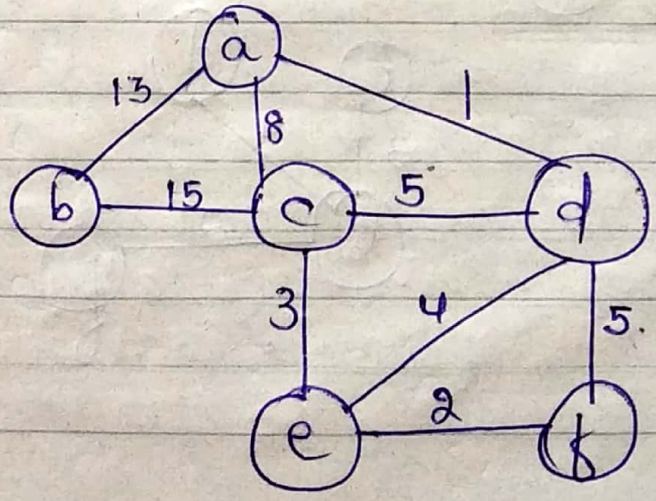
21/8/18

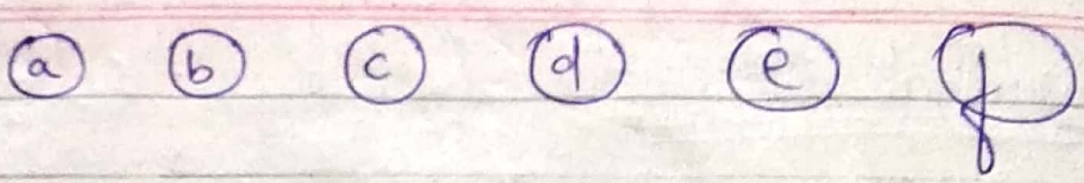
Kruskal Algo :-

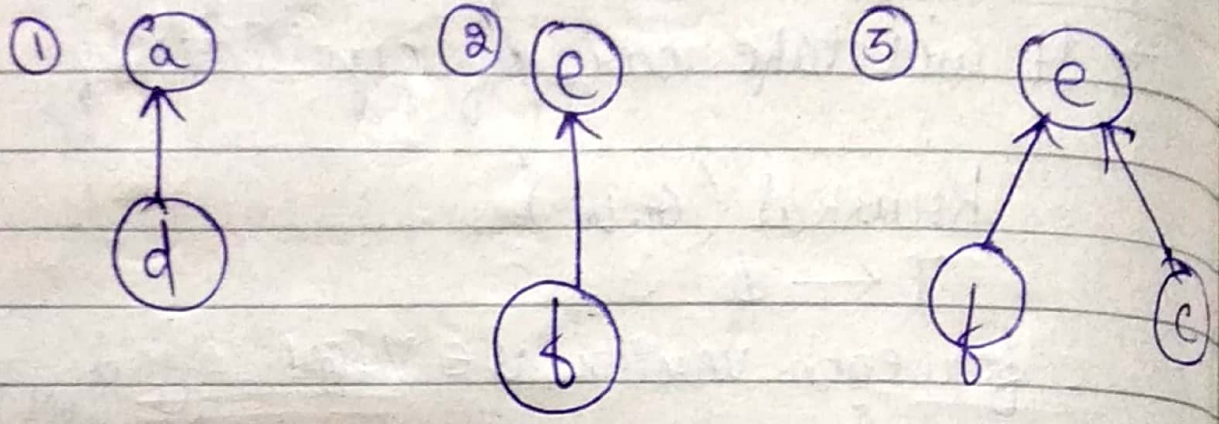
→ It will take care of cycle itself.

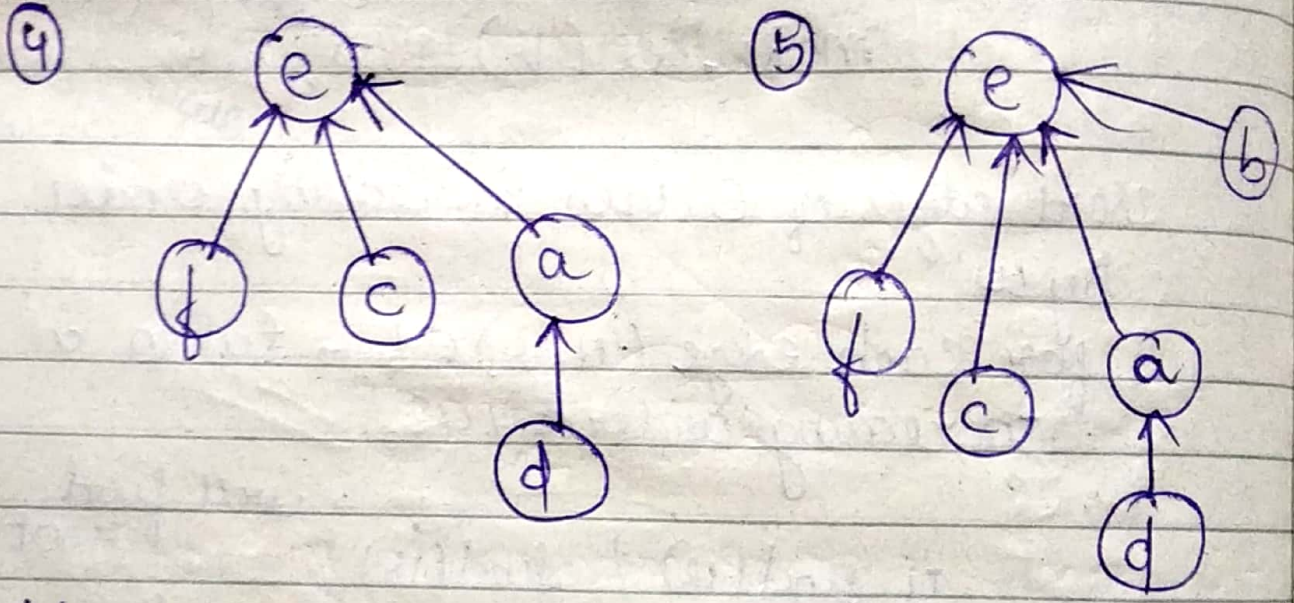
```
kruskal (G, w)
  T ← φ
for each vertex v ∈ V[G]
    {
        make_set (v)  ← create a parent array for each vertex and store -1 in each.
    }
sort edges of E into increasing order by w
for each edge (u, v) ∈ E, taken in increasing order by w
  {
    if find(u) ≠ find(v)   → will find root
    then T ← T ∪ {(u, v)}
        Union (find(u), find(v))
  }
return T
```

(1)

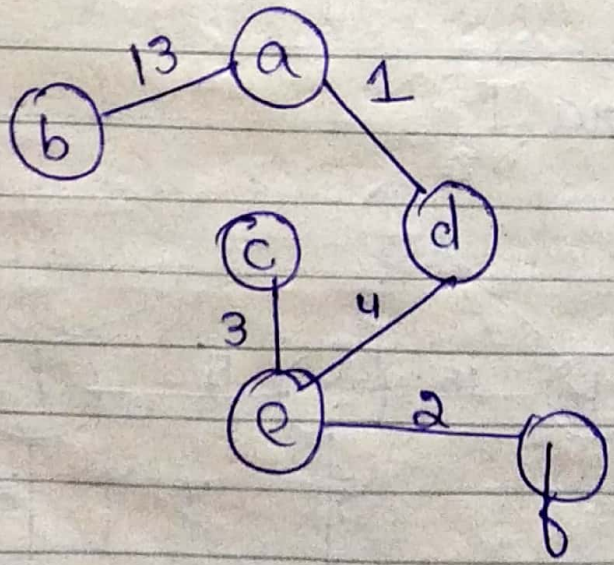




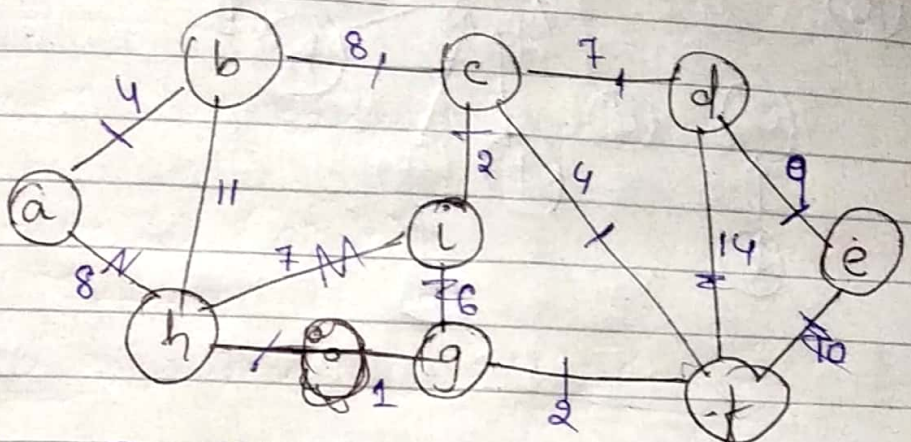




Minimum spanning tree :



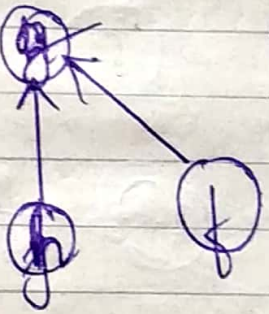
(2)



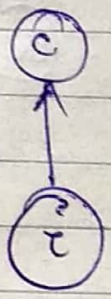
a	b	c	d	e	f	g	h	i
-1	-1	-1	-1	-1	-1	1	-1	-1

(1)

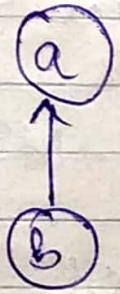
(2)



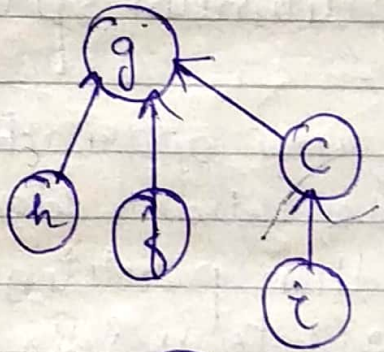
(3)



(4)



(5)



(6)

g i

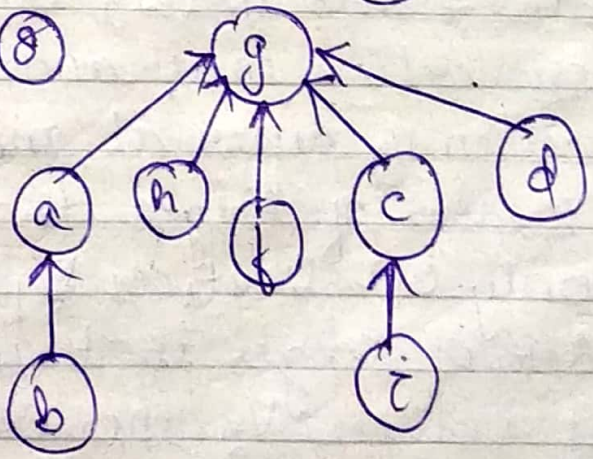
g = g

(7)

g = g

h i

(8)

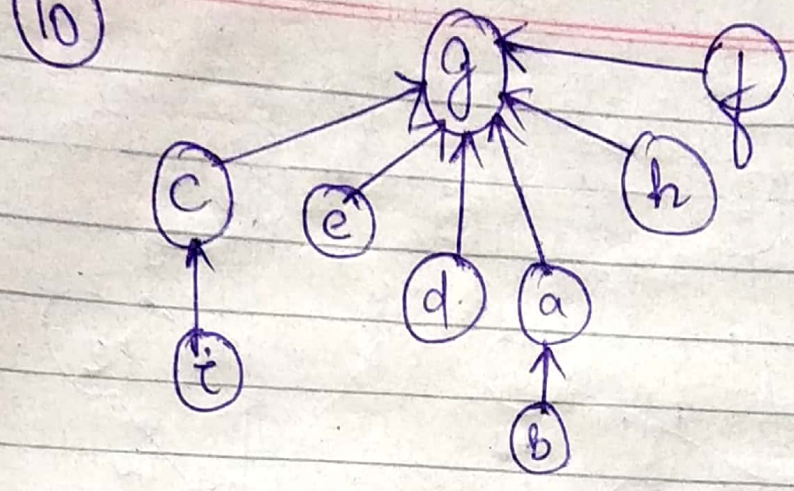


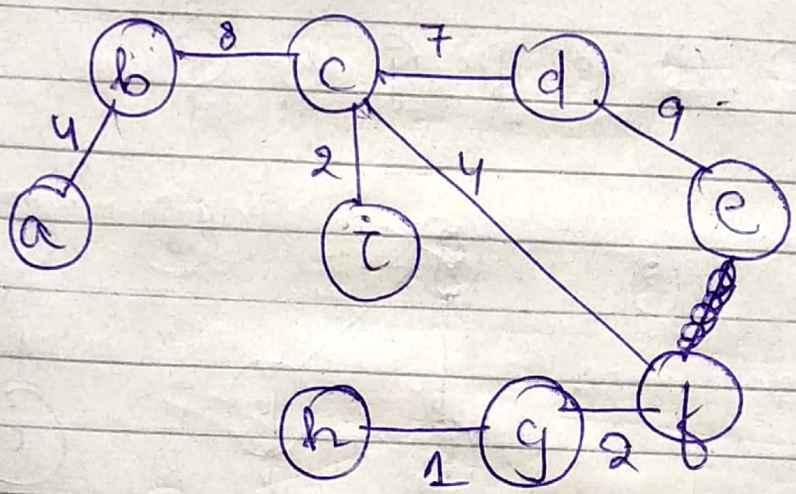
(9)

a h

g = g

(10)





Weight of minimum spanning tree = 37.

Q Divide and conquer approach to find MST.

Given $G_1(\vec{V}, E)$ → set of vertices & edges then partition set V of vertices into 2 sets V_1 & V_2 such that number of vertices in V_1 and number of vertices in V_2 differ at most by 1. Let E_1 be the set of edges that are incident only on vertices V_1 and let E_2 be the set of edges that incident only on set of vertices V_2 then recursively

solve an MST problem that each of the two subgraph which are $G_1(V_1, E_1)$, $G_2(V_2, E_2)$ finally select the min weight edge in E that crosses the cut (V_1, V_2) and use this 'e' to unite the resulting 2 MSTs into a single spanning tree.

28/8/18

Dijkstra's Algorithm:- (G, w, s).

```
Dijkstra's Algo (G, w, s)
  Initialize single-source (G, s)
   S ← ϕ
   Q ← V[G]
   while Q ≠ ϕ
    {
        do u ← Extract-min(Q)
         S ← S ∪ {u}.
        for each vertex v ∈ Adj[u]
            do Relax (u, v, w)
  }
```

```
Initialize Single Source (G, S)
  for each vertex v ∈ V[G].
    {
       do d[v] ← ∞
```

$\pi[v] \leftarrow nil$

$d[s] \leftarrow 0$

}

Relax (u, v, w)

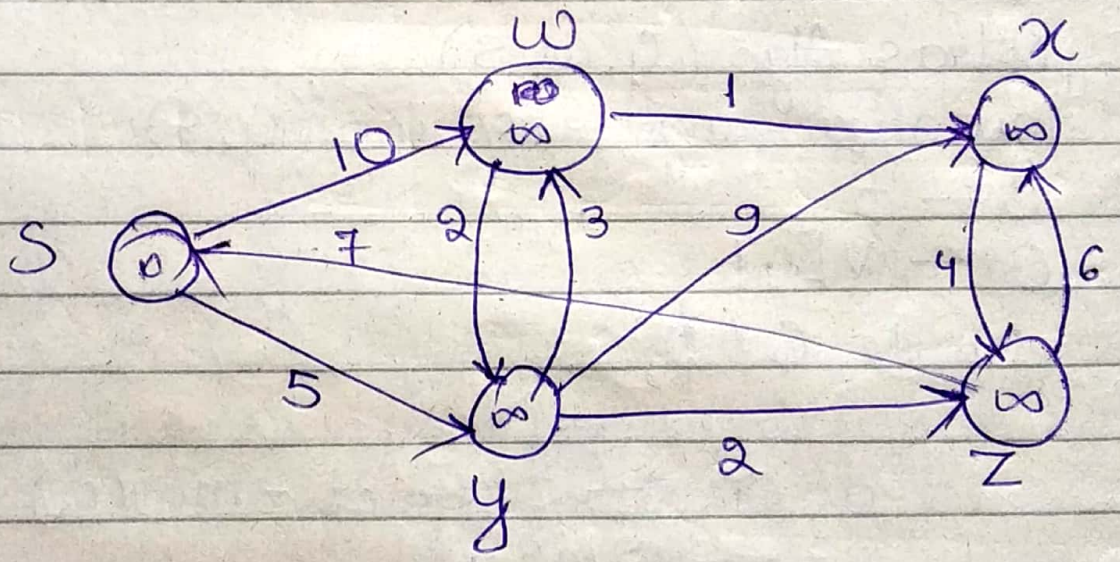
if $d[v] > d[u] + w(u,v)$

{

then $d[v] \leftarrow d[u] + w(u,v$

$\pi[v] \leftarrow u$

}



$d[v]$ → for each vertex v we have a $d[v]$ which is an upper bound on weight of shortest path from s to v and is called as a shortest path estimate.

$\pi(v)$ → predecessor array

	s	w	x	y	z
d[v] =	0	∞	∞	∞	∞

	s	w	x	y	z
π[v] =	nil	nil	nil	nil	nil

Q = S, w, x, y, z.

(1) u = S.

v = w, y

Relax (u, v, w)

For w

∞ > 0 + 5

d(w) = 5. π(w) = s

∞ > 0 + 10

d(y) = 10. π(y) = s

	s	w	x	y	z
Now d[v] =	0	10	∞	5	∞
π[v] =	nil	s	nil	s	s

(2) Now next min = y.

u = y

v = w and z and x.

10 > 5 + 3 (✓). d(w) = 8; π(w) = y

nil y w s y

For z.

$\infty > 5+2$ (✓).

$d(z) = 7$; $\pi(z) = y$.

For x.

$\infty > 5+9$ (✓)

$d(x) = 14$; $\pi(x) = y$.

	s	w	x	y	z
d[v] :	0	8	14	5	7

	s	w	x	y	z
π[v] :	nil	y	y	s	y

(3) Next min = z.

Adjacency list = x and s.

$d[v] > d[u] + w(u,v)$

$\Rightarrow 14 > 7+6$ (✓).

$d(x) = 13$.

$\pi(x) = z$.

$\Rightarrow d[s] > d[z] + w(s,z)$

$\Rightarrow 0 > 7 + 7 \quad (X)$

No this will happen.

$d[v]$:

s	w	x	y	z
0	8	13	5	7

$\pi[v]$:

s	w	x	y	z
nil	y	z	s	y

(4) Next min = w.

Adjacency list (v) = y, x.

$\Rightarrow d[v] > d[u] + w(u,v)$

$\Rightarrow 5 > 8 + 2 \quad (X)$.

$\Rightarrow d[v] > d[u] + w(u,v)$.

$\Rightarrow 13 > 8 + 1 \quad (\checkmark)$.

$\pi(x) = w$

$d(x) = 9$.

$d[v]$:

s	w	x	y	z
0	8	9	5	7

	s	w	x	y	z
$\pi[v]$ =	nil	y	w	s	y

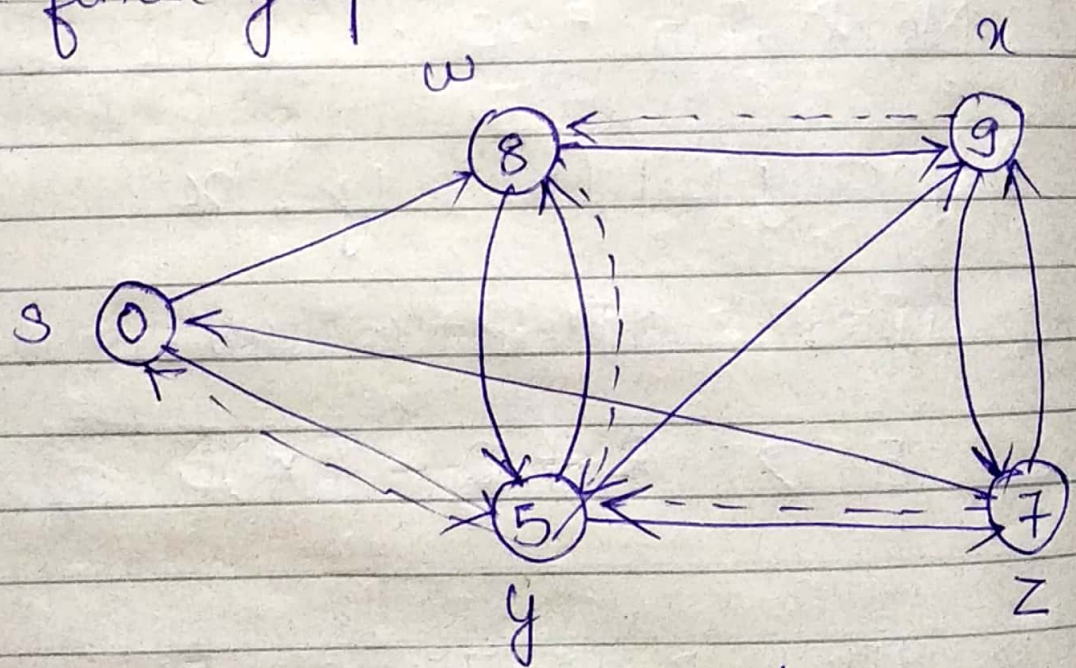
(8) next min is x.

$v = z$.

$d(z) > d(x) + w(x, z)$

$\Rightarrow 7 > 9 + 4 \ (x)$.

Our final graph is :—



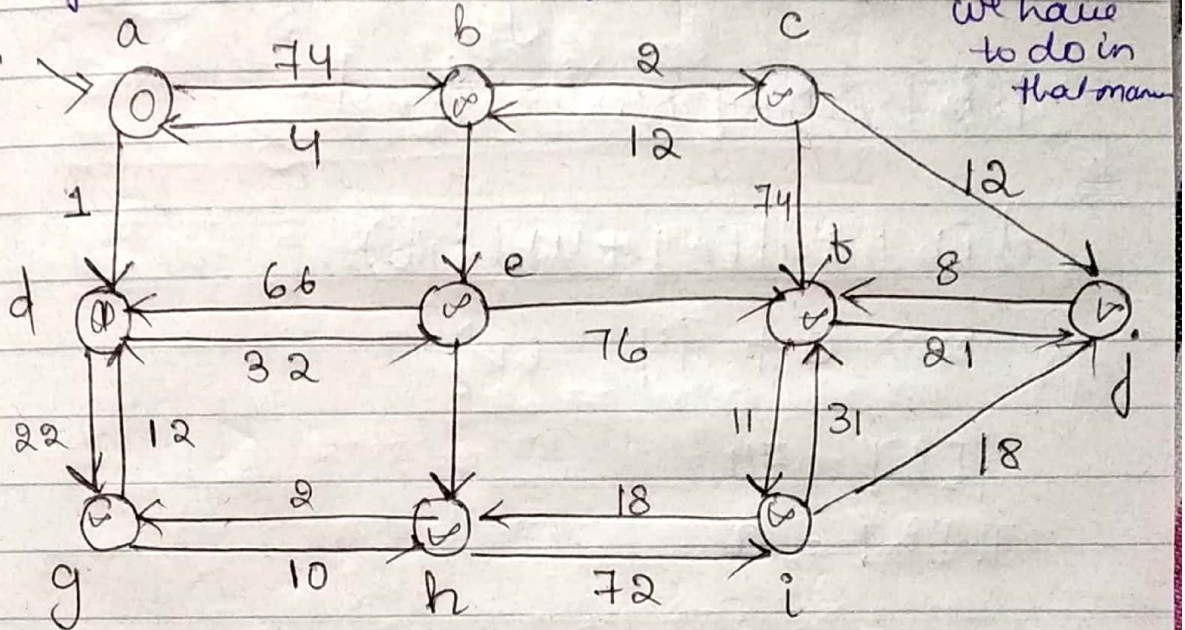
Our min path b/w s and x is s, y, w and x

Path length = $5 + 3 + 1 = 9$.

→ As it doesn't matter because we always take the min. key.

* We can take Q is any order. But if it is given that you have to take in alphabetical order then you have to take in that order.
* If doesn't matter how we extract the adjacency list but if given alphabetical then we have to do in that manner.

Q. =>



Answer.

	a	b	c	d	e	f	g	h	i	j
d[v] =	0	74	76	1	33	96	23	33	105	88

	a	b	c	d	e	f	g	h	i	j
π[v] =	nil	a	b	a	d	j	d	g	h	c

Q Assignment :- Take any wt -ve and check whether this algo work or not.

Sol.

a)

Q = b, c, d, e, f, g, h, i, j.

u = a.

(i)

(b) Adjacency list of a

V = b, d.

b

d[b] > d[a] + w(a, b)

∞ > 0 + 74 (✓)

d[b] = 74

π[b] = a.

d

d[d] > d[a] + w(a, d)

⇒ ∞ > 0 + 1 (✓)

d[d] = 1

π[d] = a.

d[v] =	a	b	c	d	e	f	g	h	i	j
	0	74	∞	1	∞	∞	∞	∞	∞	∞

π[v] =	a	b	c	d	e	f	g	h	i	j
	nil	a	nil	a	nil	nil	nil	nil	nil	nil

② min = d.

Adjacency list of d; v = g, e.

g

$d[g] > d[d] + w(g,d)$

$\Rightarrow \infty > 1 + 22$ (✓)

$d[g] = 23$

$\pi[g] = d$

e

$d[e] > d[d] + w(e,d)$

$\Rightarrow \infty > 1 + 32$

$d[e] = 33$

$\pi[e] = d$

$d[v]$ =	a	b	c	d	e	f	g	h	i	j
	0	74	∞	1	33	∞	23	∞	∞	∞

$\pi[v]$ =	a	b	c	d	e	f	g	h	i	j
	nil	a	nil	a	d	nil	d	nil	nil	nil

③ next min = g.

v = h, d.

$d[h] > d[g] + w(g,d)$

$\infty > 23 + 10$ (✓).

$d(h) = 93$

$\pi(h) = g$

$d[d] > d[g] + w(g,d)$

$1 > 93 + 2$ (X)

	a	b	c	d	e	f	g	h	i	j
d[v] =	0	74	∞	1	33	∞	23	33	∞	∞
π[v] =	nil	a	nil	a	d	nil	d	g	nil	nil

(4) next min = h.

$v = i$

$d[i] > d[h] + w(i,h)$

$\infty > 93 + 72$

$= 105$

$d[i] = 105$

$\pi[i] = h$

	a	b	c	d	e	f	g	h	i	j
d[v] =	0	74	∞	1	33	∞	23	33	105	∞
π[v] =	nil	a	nil	a	d	nil	d	g	h	nil

(5) next min = e.

$v = f, h, d$.

$\infty > 33 + 76$

$= 109$

$d[f] = 109$.

$\pi[f] = e$.

$23 > 33 + x$ (X)

$1 > 33 + x$ (X)

	a	b	c	d	e	f	g	h	i	j
d[v] =	0	74	∞	1	33	109	23	33	105	∞

	a	b	c	d	e	f	g	h	i	j
π[v] =	nil	a	nil	a	d	e	d	g	h	nil

(6) next min = b.

$v = e, c, a$.

$d[e] > d[b] + w(e, a)$

$32 > 74 + x$ (X).

$d[c] > d[b] + w(c, b)$

$\infty > 74 + 2$

76.

$d[c] = 76$.

$\pi[c] = b$.

$d[a] > d[b] + w(a,b)$

$0 > 74 + x \quad (X)$

	a	b	c	d	e	f	g	h	i	j
$d(v) =$	0	74	76	1	32	108	23	33	105	∞
$\pi(v) =$	nil	a	b	a	d	e	d	g	h	nil

⑦ next min = c.

$v = j, f, b.$

$d[j] > d[c] + w(c,j)$

$= \infty > 76 + 12$

$= \infty > 88 \quad (\checkmark)$

$d[j] = 88 \; ; \pi[j] = c.$

$108 > 76 + 74$

$= 150 \quad (X).$

$74 > 76 + x \quad (X)$

	a	b	c	d	e	f	g	h	i	j
$d(v) =$	0	74	76	1	33	108	23	33	105	88
$\pi(v) =$	nil	a	b	a	d	e	d	g	h	c

(8) min = f.

u = f.

108 > 88 + 8 (✓)

d[f] = 96 ; π[f] = j.

(9) min = f

u = i and j.

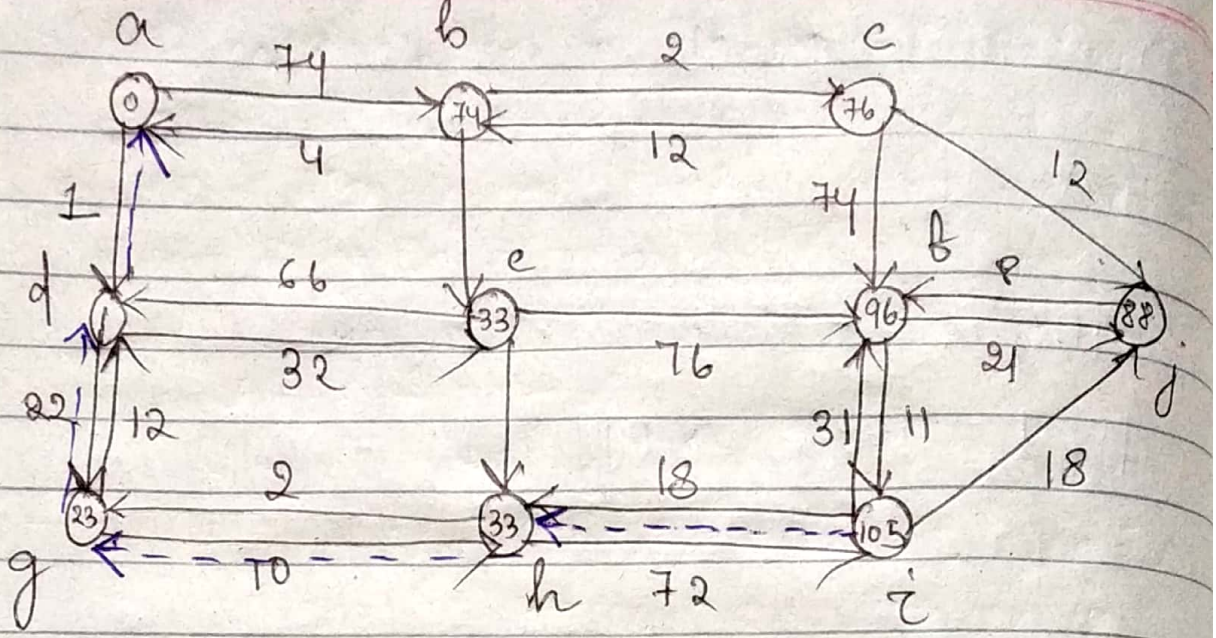
d[i] > 96 + 11

105 > 107 (X).

88 > 96 + x (X).

	a	b	c	d	e	f	g	h	i	j
d(v) =	0	74	76	1	33	96	23	33	105	88

	a	b	c	d	e	f	g	h	i	j
π(v) =	nil	a	b	a	d	j	d	g	h	c

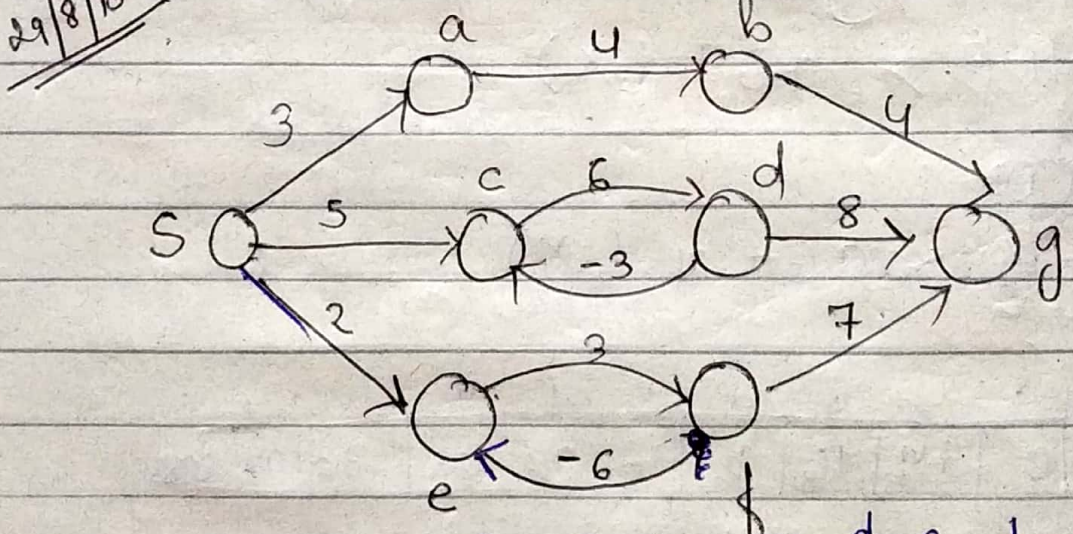


$a \to d \to g \to h \to i$

Max = 1 + 22 + 10 + 72

= (105) → Ans

29/8/18



① d[v] =

s	a	b	c	d	e	f	g
0	∞	∞	∞	∞	∞	∞	∞

Π[v] : all nil.

s a b c d e f g.

① u = s

v = c, a, e.

d[c] > d[s] + w(s,c)

∞ > 0 + 5

5 (✓).

d[c] = 5

π[c] = s.

e

∞ > 0 + 2

d[e] = 2

π[e] = s.

0	s	∞	5	∞	2	∞	∞

② u = e

v = f.

∞ > 3 + 2

d[f] = 5.

π[f] = e

③ f = 5.

v = g, e.

2 > 5 - 6 (✓)

e = -1

dijkstra fails.

so here it forms a loop. when we find shortest path b/w S & g

So here dijkstra fails. Dijkstra doesn't have any provison to check whether a -ve wt. exsist or not. So we shift to Bellman Ford algo, which have a provision to check -ve weights.

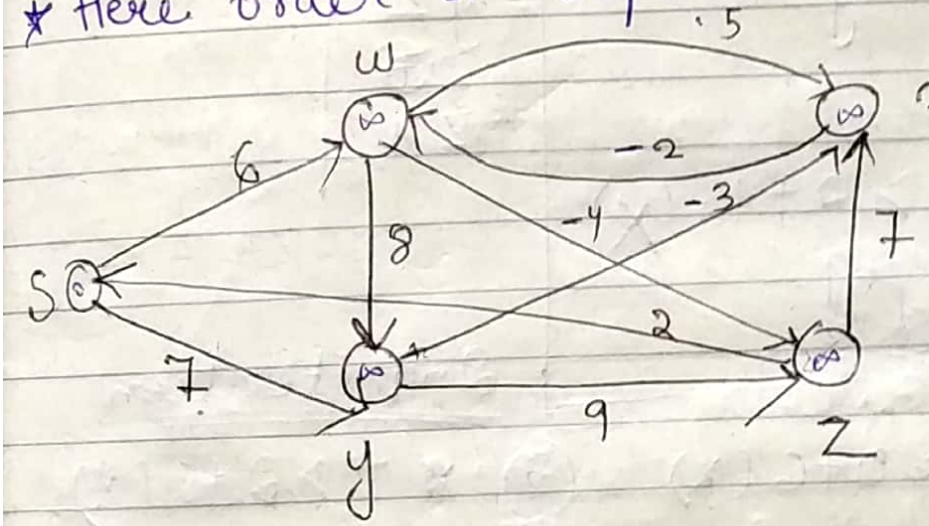
→ Dijkstra only fails when we can take one element again and again in the adjacency list (But ideally it should not be). (Confusion to be cleared later).

Bellman Ford:

```
Bellman-Ford (G, w, s)
Initialize Single Source (G, s)
    for i ← 1 to |V[G]| - 1
    {
        for each edge (u, v) ∈ E(G)
            Relax (u, v, w)
    }
for each edge (u, v) ∈ E[G]
```

{
if $d[v] > d[u] + w(u,v)$
return False.
}
Return True.

* Here order is important (as we have to relax n-1 times)



Order

~~Sy; Sw; wy, wx, wz, zx zx~~

S w x y z

1st iteration.

① Sw Sy wx xw xwy wz yx xyz xzs zx	① $\infty > 0+6$ (✓) $\pi[w] = S$ $d[w] = 6$ ② $\infty > 0+7$ (✓) $\pi[y] = S$ $d[y] = 7$	③ $\infty > 6+5$ $d[x] = 11$ $\pi[x] = w$ ④ $6 > 11$ (x) 2 ⑤ $7 > 6+8$ (x)

Assignment → Applying condition if $d[v] > d[u] + w(u,v)$ return false how you can track -ve cycle.

⑥ $\infty > 6 + (-4)$

$\infty > 2$ (✓)

$\pi(z) = w$

$d(z) = 2$

⑦ $11 > 7 + 3$ (✓)

$\pi[x] = 4$

$d[x] = y$

⑧ zx

$4 > 2 + 7$ (✗)

$d[x]$

$\pi[x]$ z

✗ sw	s - 0, nil
✗ sy	w - 6, s
✗ wx	x - 4, y
✓ xw	y - 7, s
✗ wy	z - 2, w
✗ wz	
✗ yx	
✗ yz	
✗ zs	
✗ zx	

2nd iteration

① $6 > 0 + 6$ (✗)

② $7 > 0 + 7$ (✗)

③ $4 > 6 + 5$ (✗)

④ $6 > 4 - 2$ (✓)

$\pi(w) = x$ $d(w) = 2$

⑤ $7 > 2 + 8$ (✗)

⑥ $2 > 2 - 4$ (✓)

$\pi(z) = w$; $d(z) = -2$

⑦ $4 > 7 - 3$ (✗)

⑧ $-2 > 7$ (✗)

+q

⑨ $-2 > 0 + 2$ (✗)

⑩ $4 > 2 + 7$ (✗)

Ans Q Apply Bellman-Ford on the previous -ve cycle dijkstra algo and justify.

s - 0 , nil	sw = 6 yx = -3
w - 2 , x	sy = 7 yz = 9
x - 4 , y	wx = 5 zs = 2
y - 7 , s	xw = -2
z - -2 , w	wy = 8 zx 7.
	wz -4

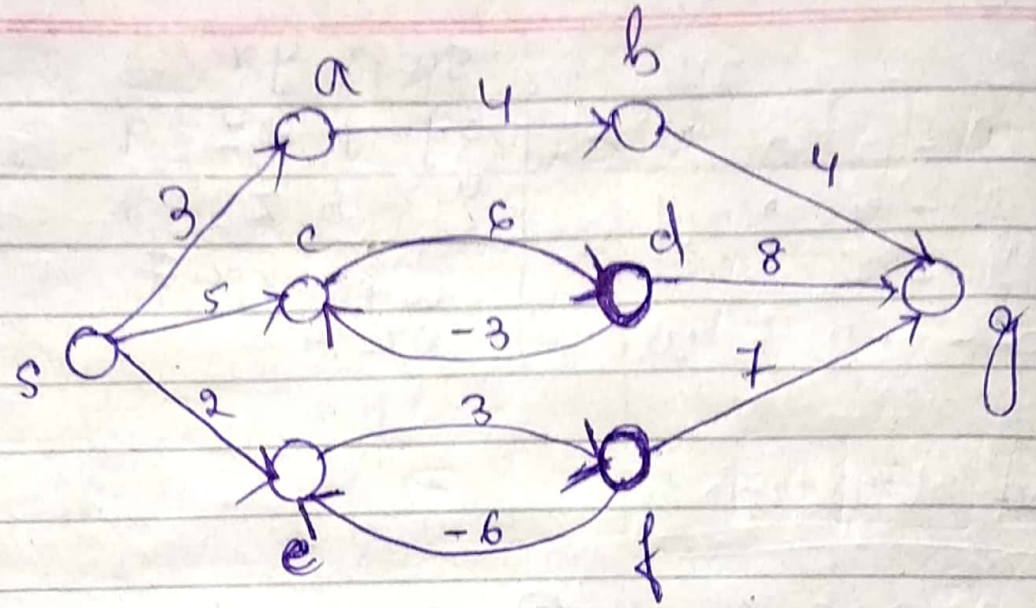
3rd iteration

① 2 > 0+6 (x)
② 7 > 0+7 (x)
③ 4 > 5+2 (x)
④ 2 > 4-2 (x)
⑤ 7 > 2+8 (x)
⑥ -2 > -4+2 (x)
⑦ 4 > 7-3 (x)
⑧ -2 > 9+7 (x)
⑨ 0 > 2-2 (x)
⑩ 4 > 7-2 (x)

→ No change in 3rd iteration.
→ So there will be no change in 4th iteration.

* Generally our final answer comes after the 2nd step (in this case). Take any order of edges.

Q Previous dijkstra algorithm:-



Just taking path from s to g via e

① u = s

v = e, c, a.

e

$\infty > 0 + 2$ (w)

$\pi(e) = s$; $d(e) = 2$

c

$\infty > 0 + 5$

$\pi(c) = s$; $d(c) = 5$

a

$\infty > 0 + 3$

$\pi(a) = s$

$d(a) = 3$

② u = e

v = f

$\infty > 2 + 3$

$d(f) = 5$; $\pi(f) = e$

③ u = f

v = e, g

e

$2 > -6 + 5$

$= -1$

$d(e) = -1$; $\pi(e) = f$

g

$\infty > 7 + 5$

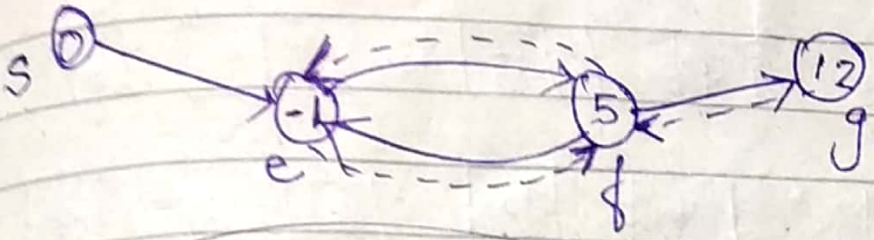
$\infty > 12$

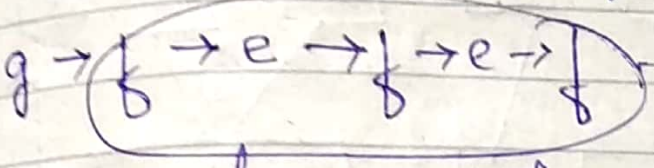
$\pi(g) = f$

$d(g) = 12$.

Now

s	e	f	g
nil.	f	e	f





trap. (There is no connection now to S).

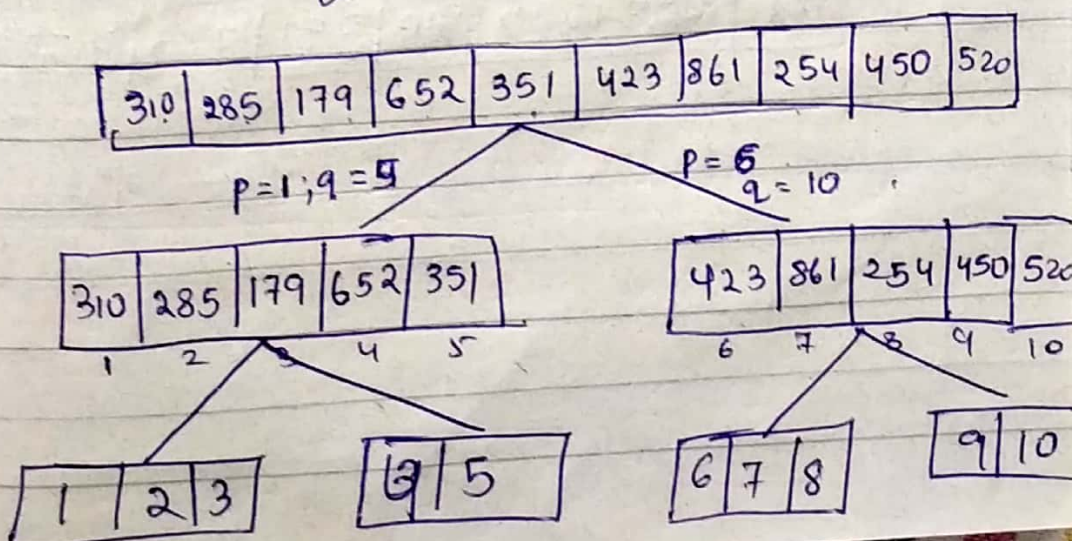
So, this is drawback of dijkstra algorithm with negative weights.

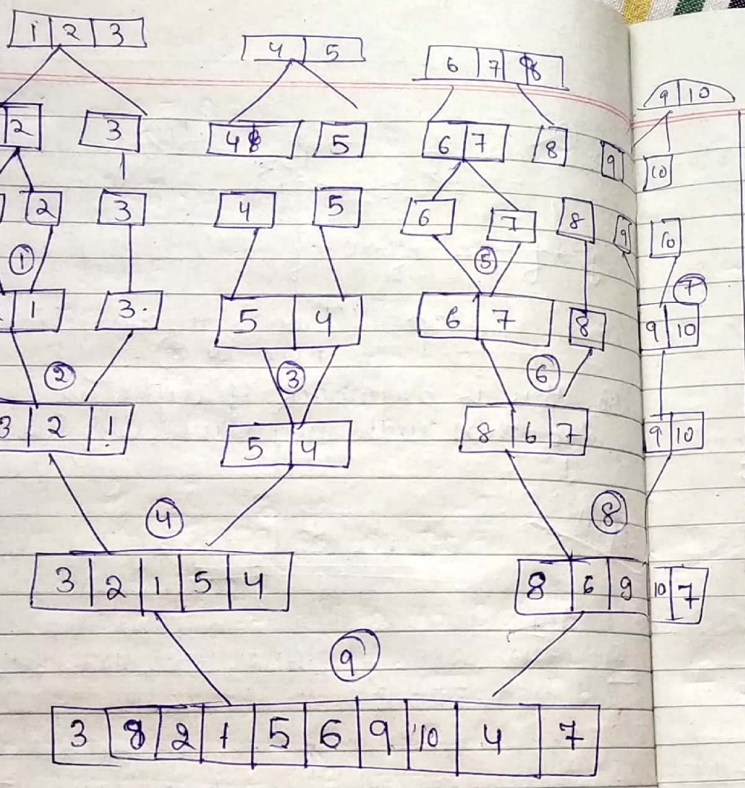
30/8/18

Tutorial-3

Q 310, 285, 179, 652, 351, 423, 861, 254, 450, 520 (1 to 10)

$$q = \frac{p+r}{2} = \frac{1+10}{2} = \frac{11}{5} = 5.$$





179	254	285	310	351	423	450	520	652	861

Q 65, 70, 75, 80, 85, 60, 55, 50, 45.

65	70	75	80	85	60	55	50	45
1	2	3	4	5	6	7	8	9

$x = 45$

$i = 0$.

① 45 | 70 | 75 | 80 | 85 | 60 | 55 | 50 | 65

② $q = 4$.

$(A, 2, 9)$.

$x = 65$

$i = 1$.

for $j = 2$ to 8

i j

60 75 80 85 70 55 50 65

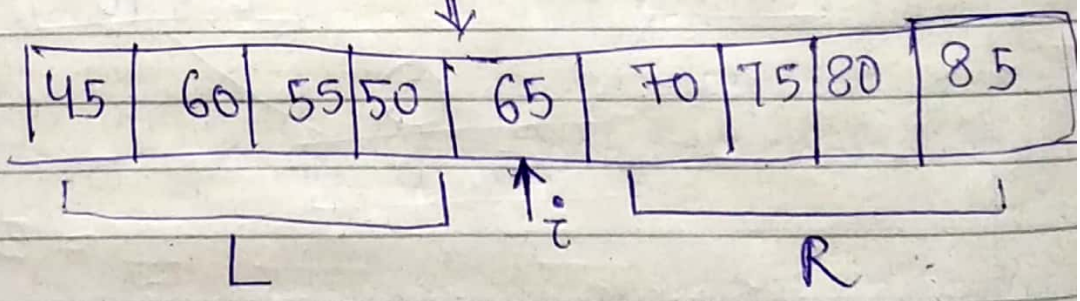
i ⇓ j

60 55 80 85 70 75 50 65

i

60 55 50 85 70 75 80 65

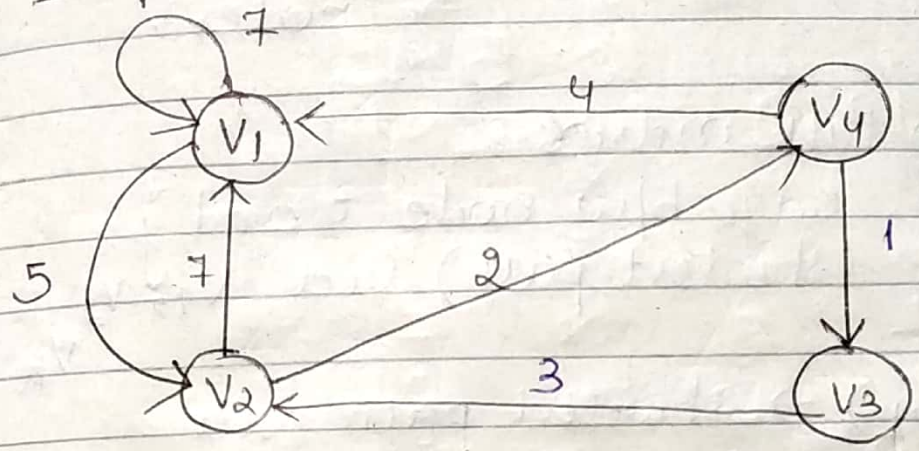
⇓



And this will keep on going first all left then all right till all is sorted.

31/8/18

All pair shortest path (Floyd Warshall)



$$Q = \begin{array}{c} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{array} \begin{array}{c} \begin{array}{cccc} V_1 & V_2 & V_3 & V_4 \end{array} \\ \begin{bmatrix} & & & \\ & & & q_{24} \\ & & & \\ & q_{42} & & \end{bmatrix} \end{array}$$

q_{42} → an edge exist b/w 4 and 2 from $4 \rightarrow 2$.

q_{24} → and edge exist b/w $2 \rightarrow 4$.

w = weight of every edge of a graph. (Weight matrix)

$$w = \begin{cases} w(e) & \text{if there exist any edge.} \\ 0 & \text{otherwise.} \end{cases}$$

$Q = q_{ij}$ (it gives shortest path b/w i and j).

$Q_0, Q_1, Q_2 \ldots Q_{M-1}$ where M is the # nodes.

$P_K(i,j)$ → Path matrix.
i.e, edge b/w node i and j (shortest path) via $V_1, V_2 \ldots V_K$

Eg $P_2(i,j)$ → shortest path via V_1 and V_2 (can be not necessary) but not V_3.

$$Q_K[i,j] = \min(Q_{K-1}(i,j), Q_{K-1}(i,k) + Q_{K-1}(k,j));$$

In Divide & conquer,
Divide your prob into sub prob and all prob are independent of each other.

In Dynamic prog, divide your problems into sub problems and all are dependent on one another.

So this algorithm is an eg. of dynamic programming.

Warshall's algo (G, m)

```
for i,j = 1 to M,
    if w(i,j) = 0 then Q0[i,j] = ∞.
    else Q0[i,j] = w(i,j).

for k = 1 to M,
{
    for i = 1 to M,
    {
        for j = 1 to M,
        {
            Qk[i,j] = min(Qk-1(i,j),
                      Qk-1(i,k) + Qk-1(k,j));
        }
    }
}
```

Initial wt vector w^2

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 7 & 5 & 0 & 0 \\ 7 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$Q_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 7 & 5 & \infty & \infty \\ 7 & \infty & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & \infty & 1 & \infty \end{bmatrix} \end{array}$$

Q_1 =

	1	2	3	4
1	7	5	∞	∞
2	7	12	∞	2
3	∞	3	∞	∞
4	4	9	1	∞

Q_2 =

	1	2	3	4
1	7	5	∞	7
2	7	12	∞	2
3	10	3	∞	5
4	4	9	1	11

Q_3 =

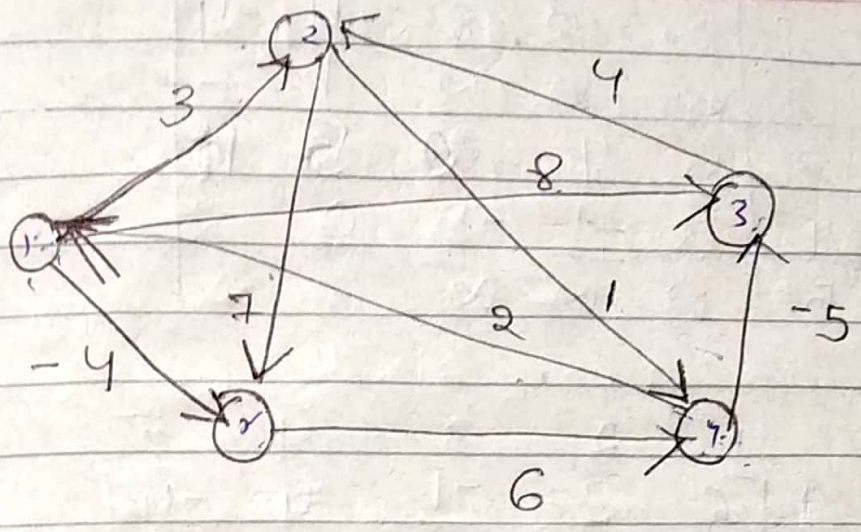
	1	2	3	4
1	7	5	∞	7
2	7	12	∞	2
3	10	3	∞	5
4	4	9	1	6

Q_4 =

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

Assignment → To modify this algorithm so that it is able to trace the path.

Q.



$$Q_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & 3 & 8 & \infty & -4 \\ \infty & \infty & \infty & 1 & 7 \\ \infty & 4 & \infty & \infty & \infty \\ 2 & \infty & -5 & \infty & \infty \\ \infty & \infty & \infty & 6 & \infty \end{bmatrix} \end{array}$$

$$Q_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & 3 & 8 & \infty & -4 \\ \infty & \infty & \infty & 1 & 7 \\ \infty & 4 & \infty & \infty & \infty \\ 2 & 5 & -5 & \infty & -2 \\ \infty & \infty & \infty & 6 & \infty \end{bmatrix} \end{array}$$

$$Q_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & 3 & 8 & 4 & -4 \\ \infty & \infty & \infty & 1 & 7 \\ \infty & 4 & \infty & 5 & 11 \\ 2 & 5 & -5 & 6 & -2 \\ \infty & \infty & \infty & 6 & \infty \end{bmatrix} \end{array}$$

Q3:

	1	2	3	4	5
1	∞	3	8	4	-4
2	∞	∞	∞	1	7
3	∞	4	[illegible]	[illegible]	[illegible]
4	2	-1	-5	0	-2
5	∞	∞	∞	6	∞

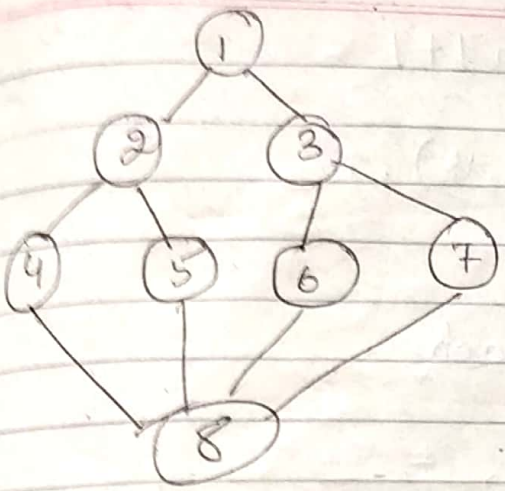
Q4:

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

Q5=

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

4/9/18



BFS

```
BFS(G,s)
for each vertex u ∈ V[G] - {s}
    do color[u] ← white
       d[u] ← ∞
       π[u] ← nil.
color(s) ← grey
d[s] ← 0
π[s] ← nil
Q ← ϕ
Enqueue(Q,s)

while Q ≠ ϕ
{
  do u ← dequeue(Q).
  for each v ∈ Adj[u]
  {
     if color[v] = white
     then color[v] ← grey
```

$d[v] \leftarrow d[u] + 1$

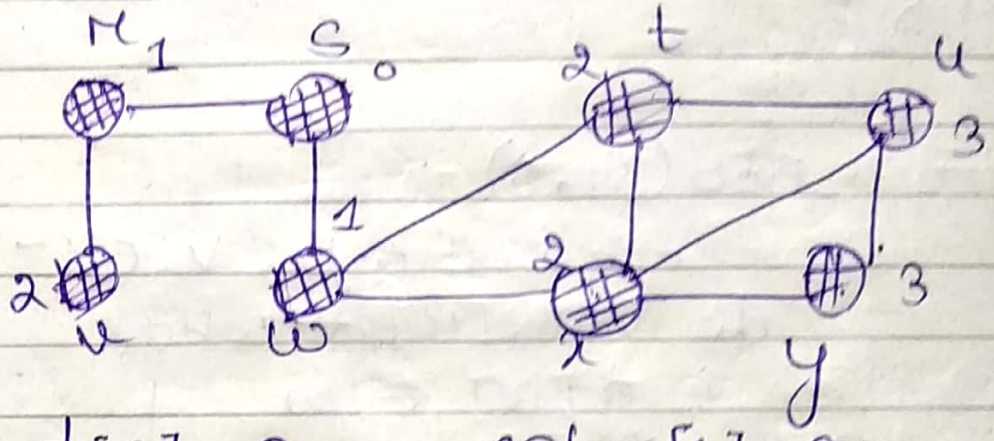
$\pi[v] \leftarrow u$

Enqueue (Q, v).

}

}

color $[u] \leftarrow$ Black.



$d[s] = 0$ color $[s] =$ grey

① $Q \leftarrow s$.

$u = s$.

Adjacency list = r, w

color $[r] \leftarrow$ grey

$d[r] = d[s] + 1$

$= 1$

$\pi[r] = s$

$Q \leftarrow r,$

color[w] ← grey

d[w] = d[s] + 1

= 1

π[w] = s.

Q ← r, w.

color[s] ← Black

② u = π

Adjacency → v and s

color[v] ← grey

d[v] = 1 + 1

= 2

π[v] = π

Q ← w, v.

color[r] = Black

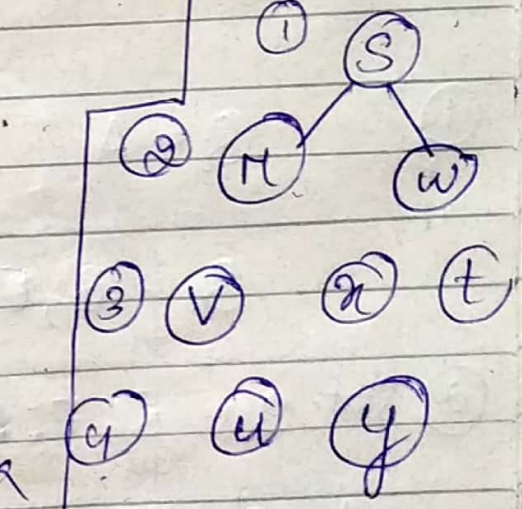
③ d[t] = 2

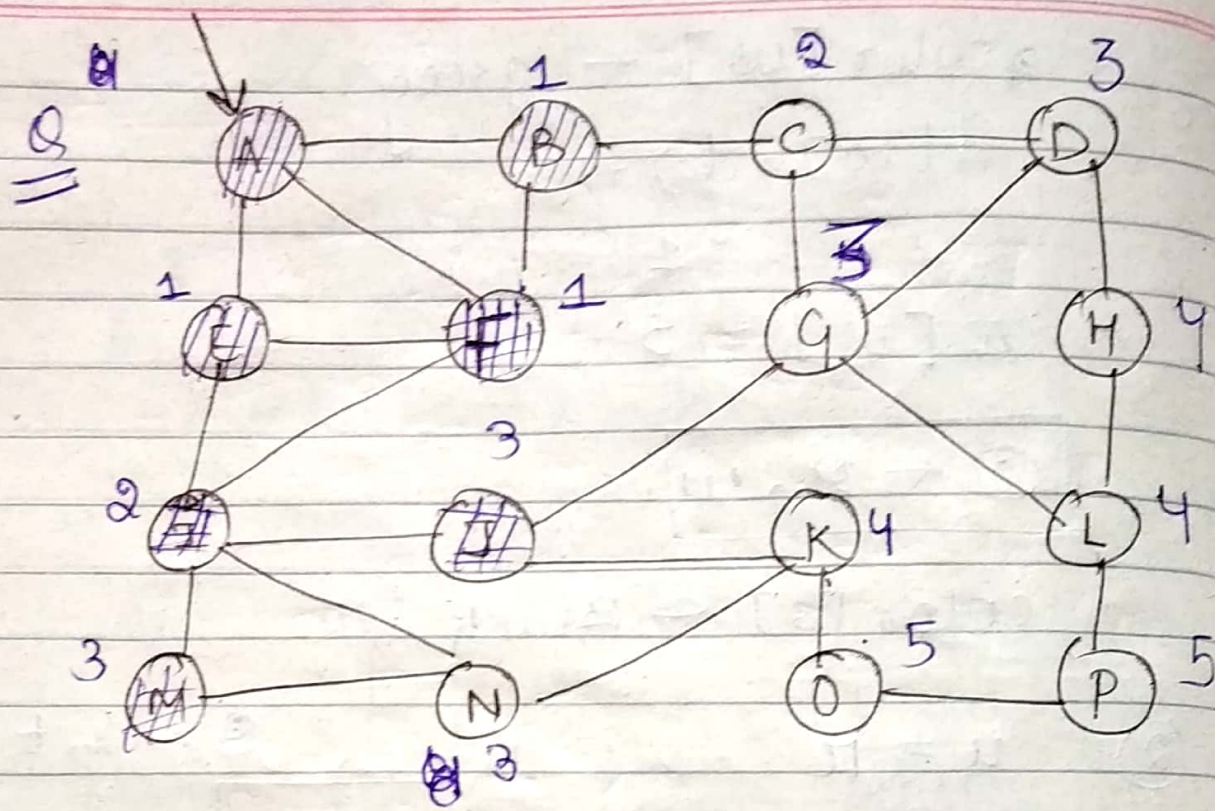
d[x] = 2

Q ← v, x, t.

Q = x, t.

Q = t, y, u





* If given alphabetical order, then the adjacency list should be in alphabetical order.

① Q = A

Q = B, E, F

② start Q = E, F,

end Q = E, F,

③ start = Q = F,

end Q = F, , I

④ Q = , I

⑤ start Q = ϕ

end Q = J, M

⑥ start Q = M

end Q = M, G, K

(7) Start = G, K

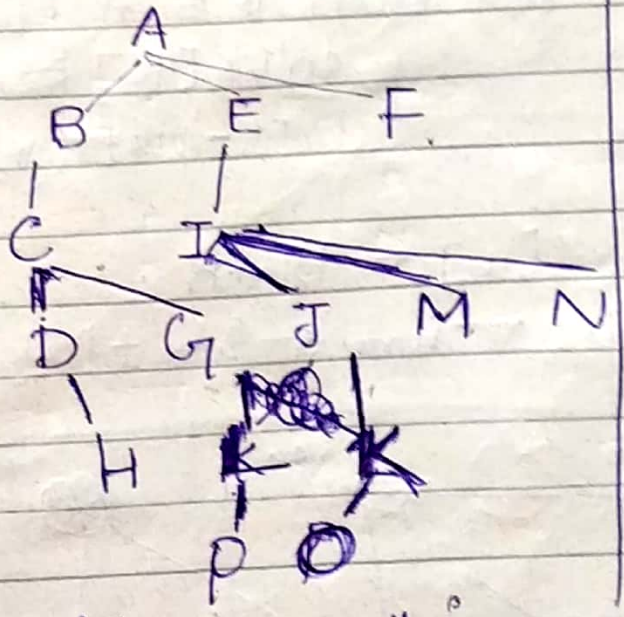
end S = G, K, N.

A B C D E F G H

I J K L M N O P

A B E F C O E I D G J O M N

H L K P O



Strongly connected ??

BFS tree for this graph.

* Here tree is according to how you have taken the element in the queue

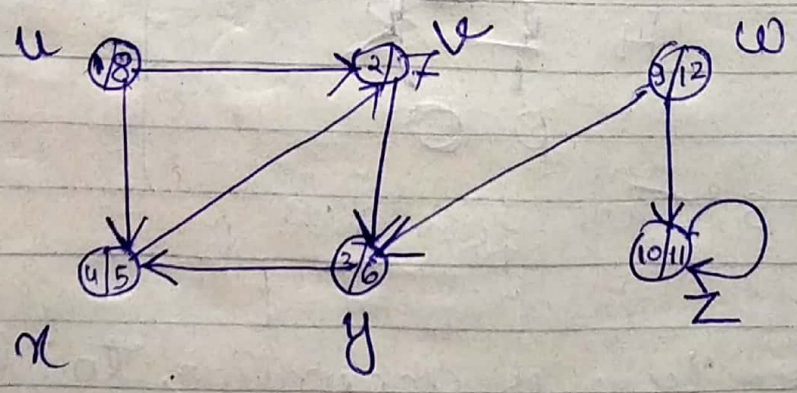
7/9/18

DFS.

```
DSF(G)
for each vertex u ∈ V[G].
    Color[u] ← white.
time ← 0
for each vertex u ∈ V[G]
  do if color[u] = white
        then DFS-Visit(u).

DFS-visit(u)
  Color[u] ← grey
  time ← time + 1.
  d[u] ← time
      for each v ∈ Adj[u]
          if color[v] ← white
            DFS-vist(v).

Color[u] ← black.
f[u] ← time ← time + 1.
```



$u = vx$

$v = y$

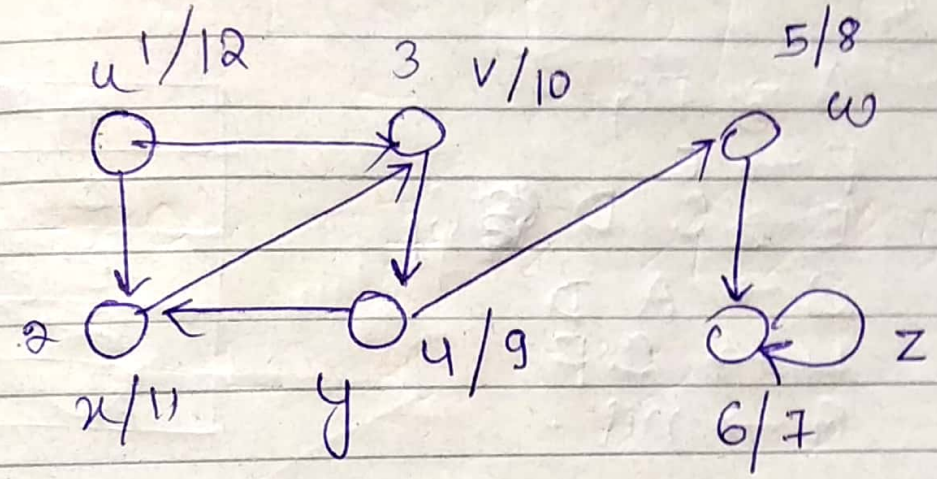
$y = x$

$x = v$

$w = yz$

$z = z$

Q



$u = xv$

$v = y$

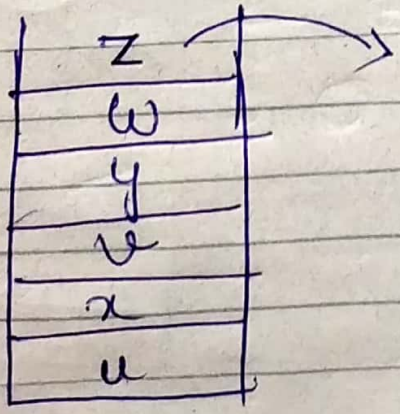
$y = xw$

$x = v$

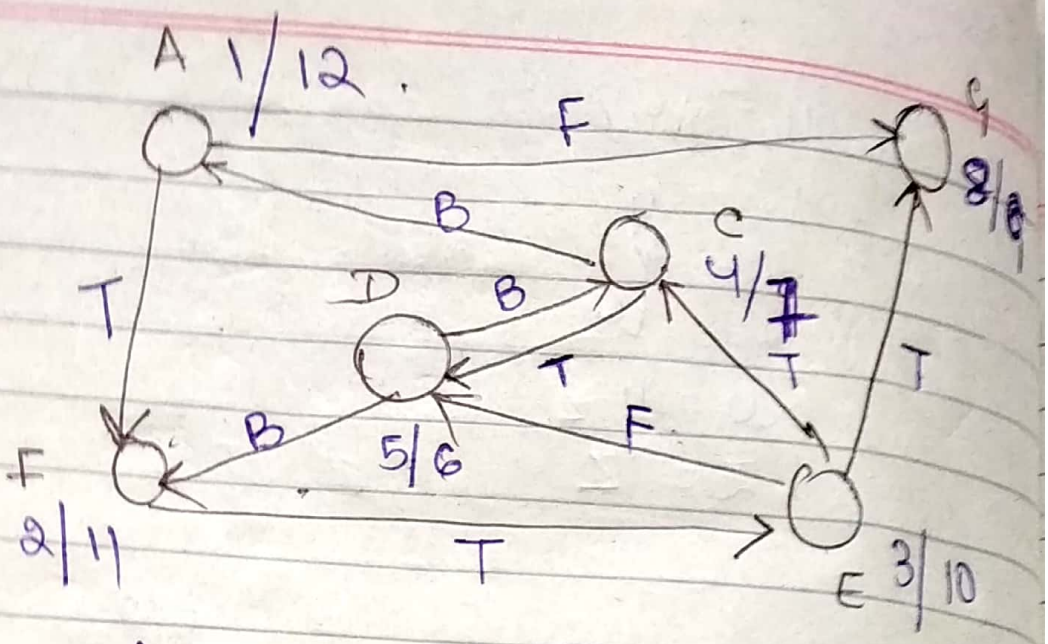
$w = yz$

$z = z$

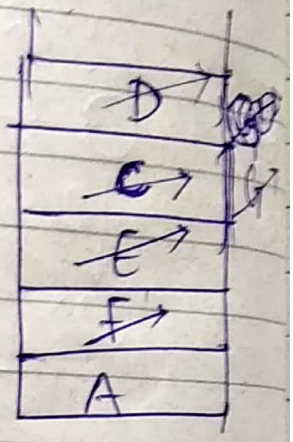
First

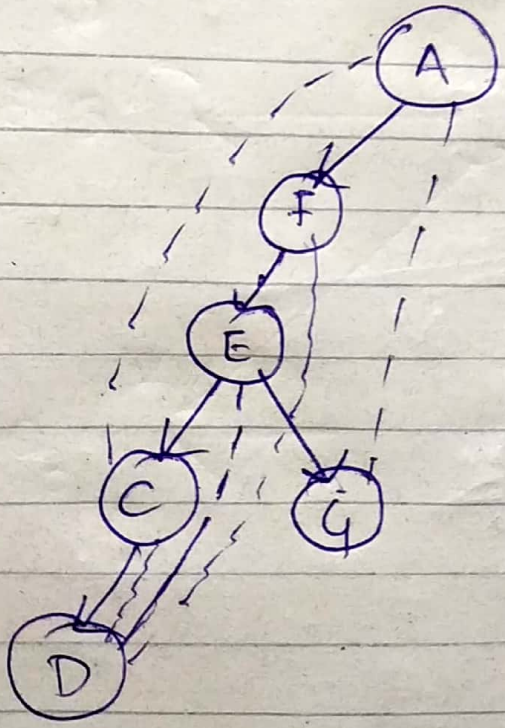


Q

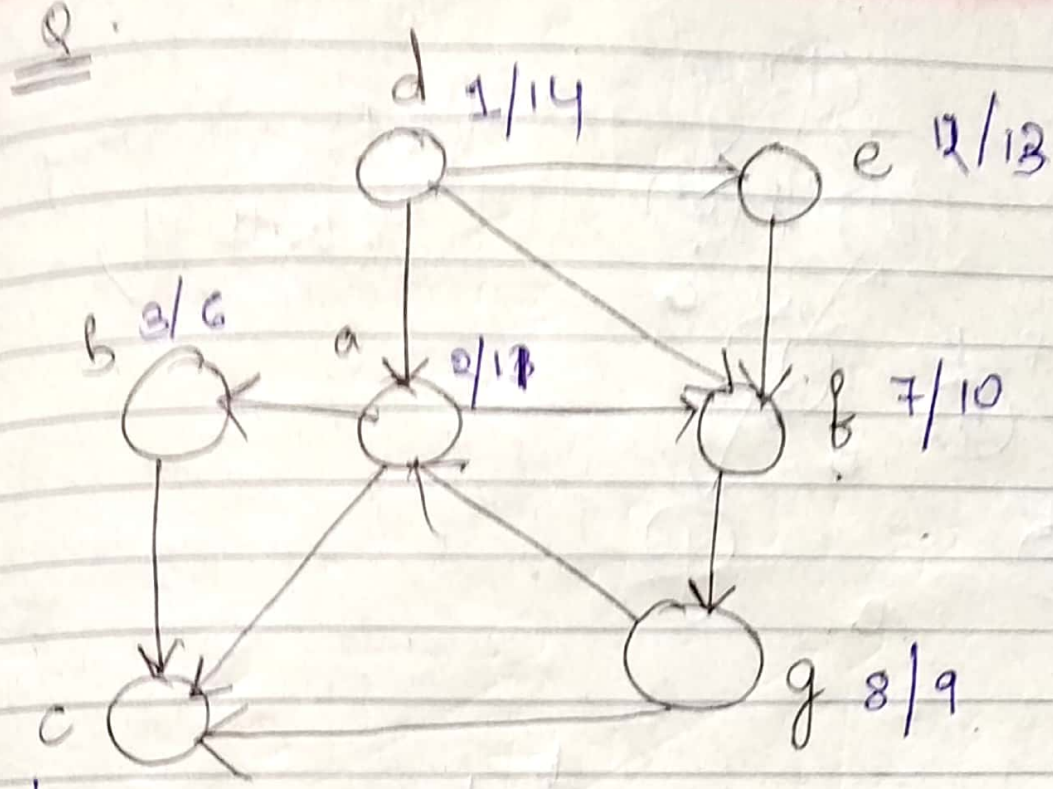


A → FG
F → E
E → C D G
C → A D
D → C F
G → nil.



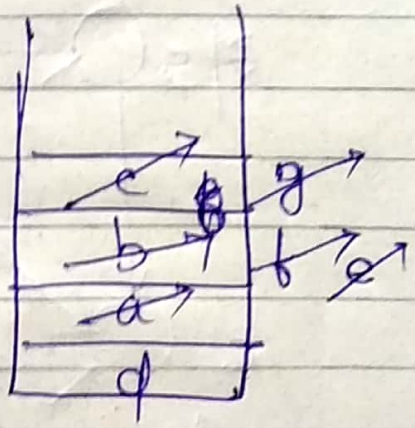


Q.



Start with d and take adjacency list in alphabetical order.

d → a, e, f
a → b, c, f
b → c
c → nil
f → g
e → f
g → a, e

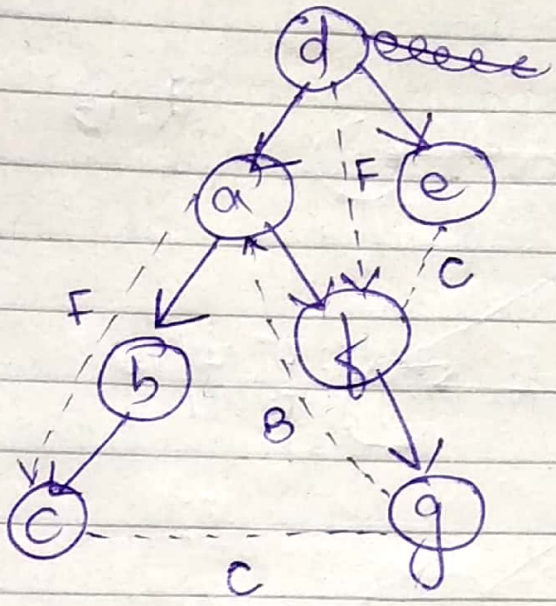


Now starting with a.

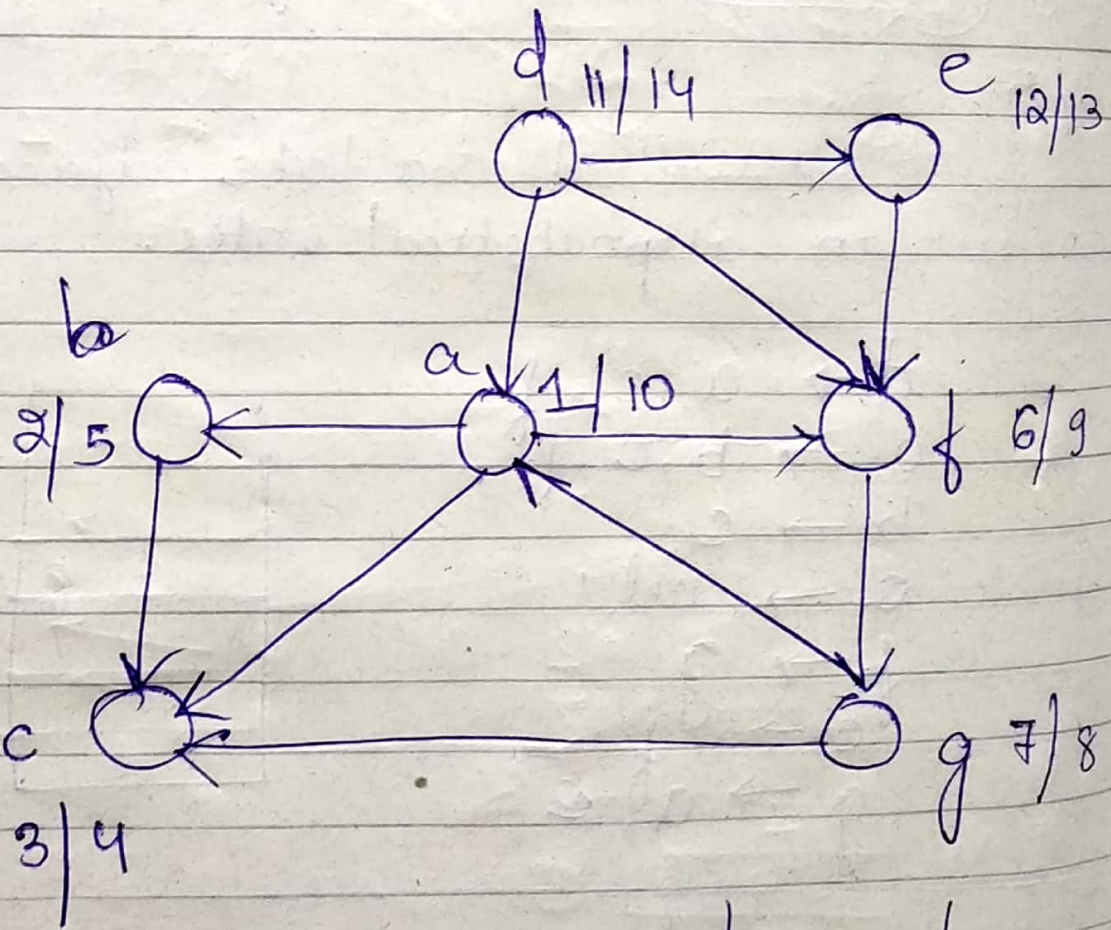
a → b, c, f
b → c
c → nil.

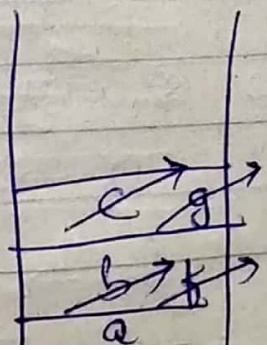
f → g
g → a, c

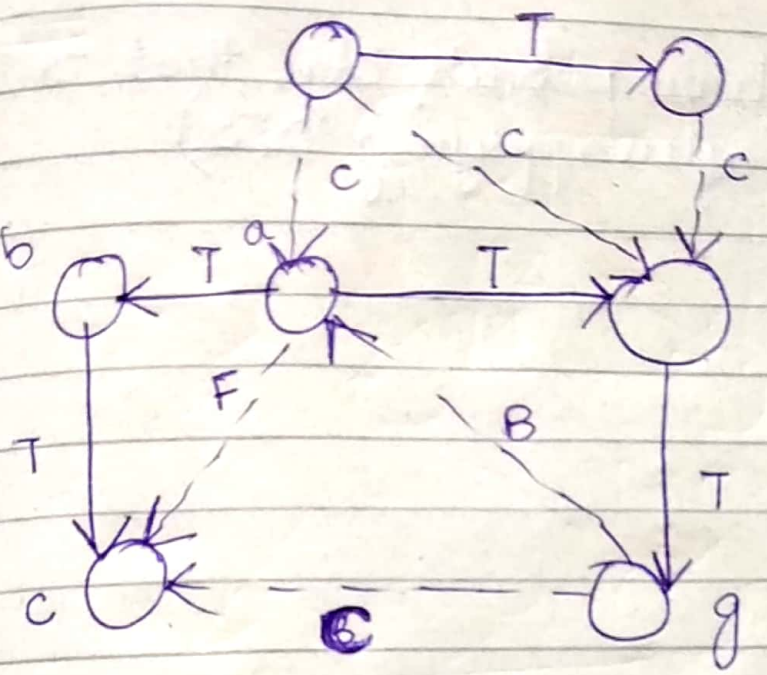
d → a e f
e → f



(DFS starting with d).







Tree edge :- edges that are in DFS tree

forward edge :- connect descendants from the ancestors.

(u, v) :- forward edge is u to descendant v.

backward edge :- vice-versa of forward edge.

Cross edges :- all the edges b/w 2 different trees.

→ Ancestors & Decendants relationship defined related to root.

(that are neither ancestors nor descendants have CE)

→ The graph having back edge that is a cycle. (after applying DFS)