time Appointment Notes

Graph

Abstract Data Type graph

{

instances: a graph is collection of vertices and edges

operations:

create() - to create a graph

display() - to display a graph

}

A graph G is collection vertices V and edges E.

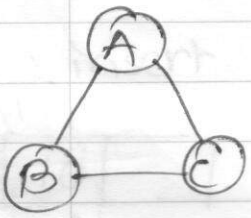
where both V & E are finite set and V is non-empty set.

Vertices are nodes in graph which holds the value, where as edges connects two adjacent vertices.

∴ G = { V, E }

ede.

G =



V = { A, B, C }

E = { AB, AC, BC }

G = { V, E }

Notes | Appointment time

Graph V/s Tree (both are non-linear data structure)

Graph (G)	Tree (T)
1) it is collection of vertices & edges	1) it is collection of nodes & branches
2) each ~~node~~ vertex can have any number of edges	2) each node can have any number of branches for general tree but for binary tree almost 2 e branches.
3) no unique vertex called as root	3) unique node called as root, start of tree
4) there can be cycle in G	4) there can not be cycle in T
5) to find shortest path in graph,	5) for decision tree, game tree, search tree

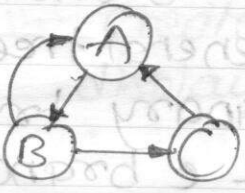
time Appointment Notes

Types of graph and graph properties

Type 1) Directed graph

In this directions are on edges indicating from one vertexe to another vertexe. So one edges one direction only; not vice versa

ede.



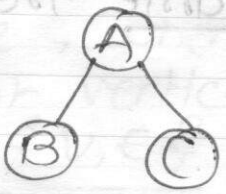
V = { A, B, C }

E = { AB, BA, BC, CA }

Type 2) Undirected graph

In this directions are not present on any of the edge. So writing from both way is allowed.

ede.



V = { A, B, C }

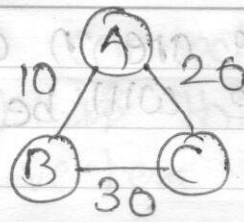
E = { AB. or BA, AC or CA }

Type 3) weighted graph

In both directed and undirected graph if all edges contains weight (numeric value associated with it) then such a graph is called as weighted graph.

Notes Appointment time

eg:

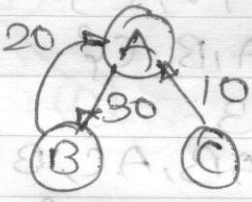


$G = \{V, E\}$

$V = \{A, B, C\}$

$E = \{ AB(10), AC(20), BC(30) \}$

ex_2



$G = \{V, E\}$

$V = \{A, B, C\}$

$E = \{ AB(30), BA(20), CA(10) \}$

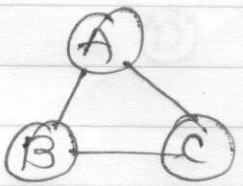
prop 1)

Type

4) complete graph - in graph G if from every vertex to every other vertex there as an edge.

∴ for undirected complete graph G_u if there are n vertices then there will be $\frac{n * n-1}{2}$ no of edges.

eg.



here $n = 3$ no of vertices

∴ no of edges $= \frac{n*(n-1)}{2} = \frac{3(2)}{2} = 3$

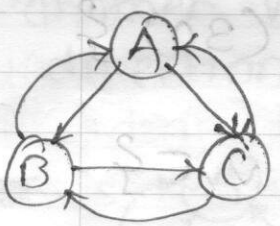
∴ G is complete graph

time Appointment Notes

complete
for directed graph if there are n of vertices in graph then there will be

n*(n-1) no of edges.

eg.



here $G = \{V, E\}$
$V = \{A, B, C\}$
$\therefore n = 3$
$E = \{AB, AC, BC, BA, CA, CB\}$

$\therefore$ total no of edges in G
$= n*(n-1)$
$= 3*2 = \underline{6}$

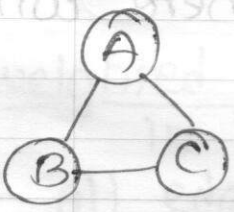
prop 2) Type 5) subgraph

a G' is subgraph of G if $G = \{V, E\}$
$G' = \{V', E'\}$

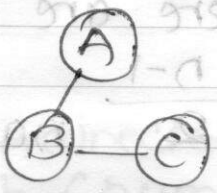
and $V' \subseteq V$ & $E' \subseteq E$

eg.

G =

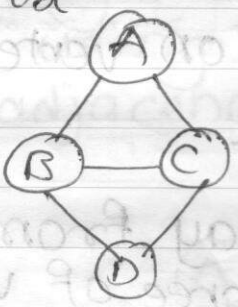


G' can be



Prop3) connected graph - in a connected graph
Type 6) if there is path from every vertex to every other vertex.

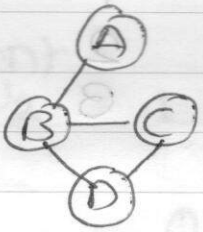
exe.



from	to	path
A	B	AB, ACB, ...
A	C	ABC, AC, ...
A	D	ABD, ACD, ...

prop4) path is sequence of edges vertex and their interconnecting edges.

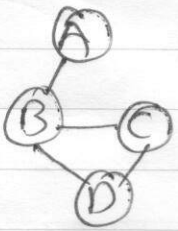
exe.



path A-B-C-D

prop5) if in a path starting vertex & ending vertex is same then such path is cycle

exe.



cycle → B-C-D-B

time Appointment Notes

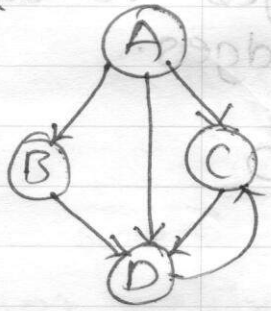
prop 6)

total no of edges associated with a vertex V is called as degree of V.

total no of edges incident on vertex V is called as indegree of V.

total no of edges going away from vertex V is called as outdegree of V.

ex.



vertex	degree	indegree	Outdegree
A	3	0	3
B	2	1	1
C	3	2	1
D	4	3	1

prop 7) self loop means an edge connects same vertex itself.

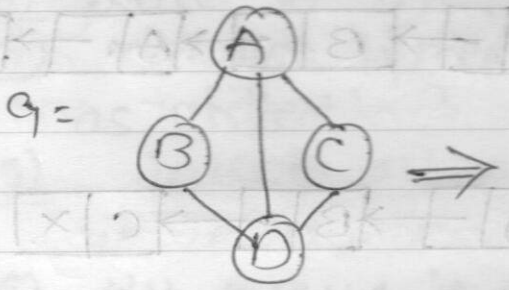
ex.

Notes | Appointment time

Representation of graph

a graph can be represented using an array called as adjacency matrixe or using a linked list called as adjacency list.

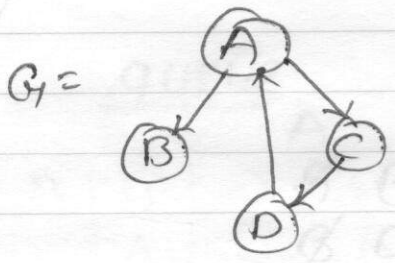
ex.



Adjacency Array matrixe

	A	B	C	D
A	0	1	1	1
B	1	0	0	1
C	1	0	0	1
D	1	1	1	0

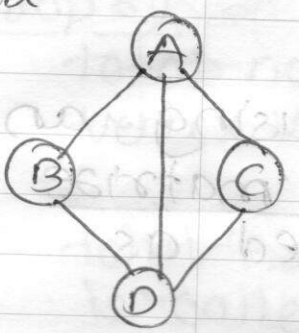
element
V_1V_2 = 1 if there is an edges from vertex V_1 to V_2
0 else zero



	A	B	C	D
A	0	1	1	0
B	0	0	0	0
C	0	0	0	1
D	1	0	0	0

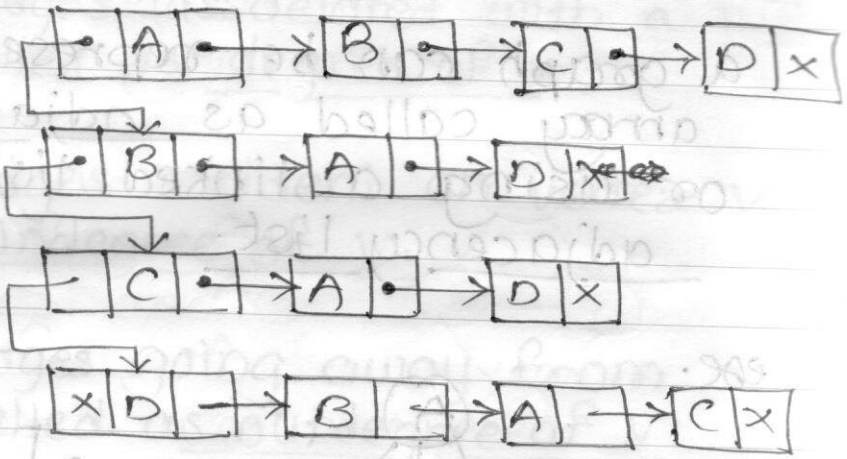
time | Appointment | Notes

8 ex.

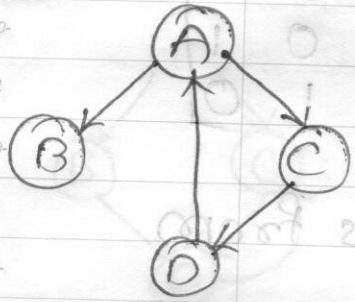


⇒

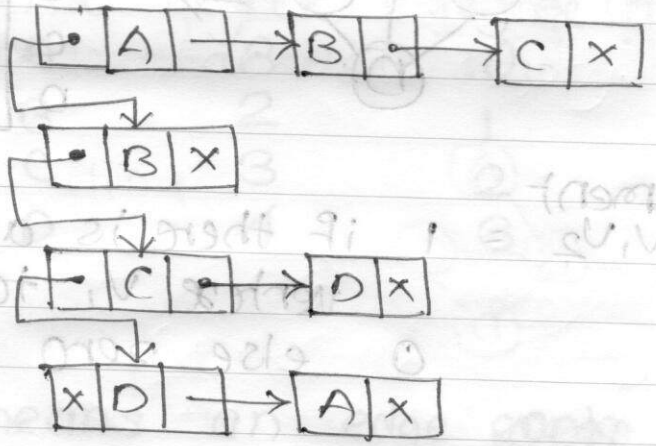
Adjacency list

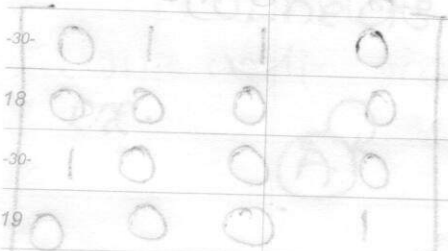


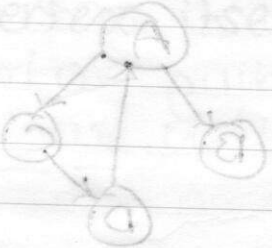
ex.



⇒







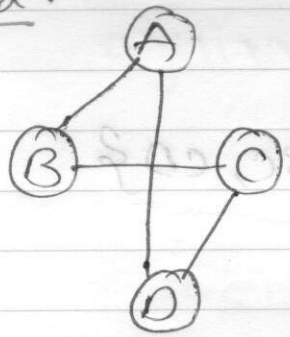
Graph traversal

1) BFS (Breadth first search)

algorithm

1) read a graph $G = \{V, E\}$
2) selected v_i from V
3) place v_i in queue
4) remove vertex from queue (declare as visited)
5) insert all non visited adjacent of it in queue
6) till queue is not empty . go to step 4

ex.



$G = \{V, E\}$

$V = \{A, B, C, D\}$

$E = \{AB, AD, BC, CD\}$

queue

A

~~A~~ B D

~~B~~ D C

~~D~~ C

~~C~~

empty

BFS: A - B - D - C

time Appointment Notes

2) DFS (Depth First search)

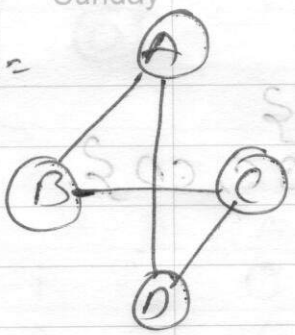
algorithm

1) read a graph $G = \{V, E\}$
2) select any v_i from V
3) push (v_i) in stack
4) read stack top and
   i) if there are any non visited adjacent nodes of it then push them in stack
   ii) else pop that v from stack and declare as visited.
5) repeat step 4 till stack not empty.

eg.

11 Sunday

$G =$



$\Rightarrow G = \{V, E\}$

$V = \{A, B, C, D\}$

$E = \{AB, AD, BC, CD\}$

Stack

	I	II	III	IV	V	
	D	B	C	B	A	empty
A	B	A	B	A		
	A		A			

DFS : D - C - B - A

Notes | Appointment | time

shortest path algorithm on graph

i) from one vertex to another
graph with only positive edges
Dijkstra's algorithm

ii) from one vertex to another
graph with -ve edges too
Bellman-Ford algorithm

iii) all pair shortest path algorithm
(with -ve edges) Floyd-Warshall algo

Applications of shortest path algorithm

time Appointment Notes

Dijkstra's algorithm:

- finds shortest path from source node to destination node
- always selects minimum weighted adjacent node, hence solves problem by greedy algorithm.
- ~~Algorithm~~ can work on directed and undirected weighted graph. (connected)
- only there should note exist any -ve edge, else fails
- Algorithm

```
function dijkstra(G, source)
{
    dist[source] = 0
    prev[source] = UNDEFINED

    for (all vertex v in G)
    {
        if(v ≠ source)
        {
            dist[v] = INFINITY
            prev[v] = UNDEFINED
            add v in Q // Q → unvisited nodes
        }
    }
```

```
while (Q != empty)
{
    u = min(dist[v] in Q)
    remove u from Q.

    for (each neighbor v of u)
    {   // unvisited neighbor
        // which is still in Q
    }
        alt = dist[u] + length(u, v)
        if (alt < dist[v])
        {
            dist[v] = alt
            prev[v] = u
        }
    }
}
return dist[], prev[]
}
```

if interested in only one target vertex then in above algorithm

break while loop

if u = target

after executing for loop.

i.e.

```
{ if (u = target)
      break while }
```

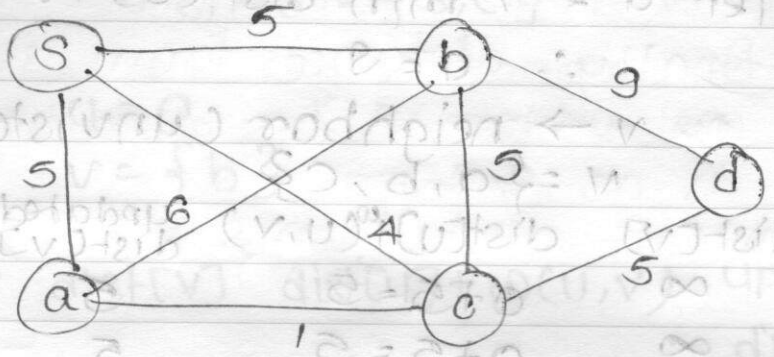
time | Appointment | Notes

8

Application of Dijkstra's algorithm

1) link state routing protocol
2) (open shortest path first (OSPF) routing protocol
3) Intermediate system to Intermediate system routing protocol
4.) practical geographic information system (GIS)
5) currency conversion problem

ex (1)



step (0) in above graph $G = \{V, E\}$

$V = \{s, a, b, c, d\}$

where s is source vertex

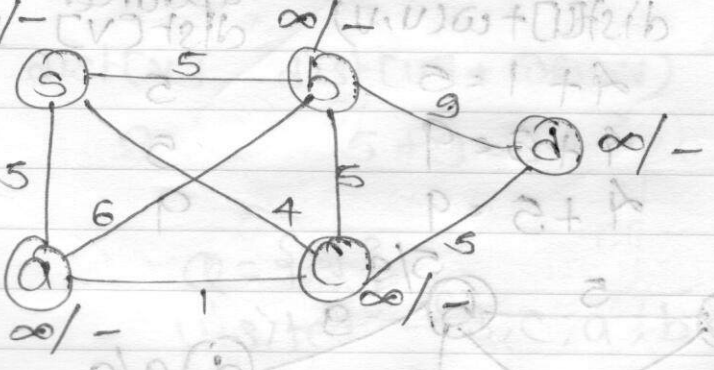
step 1)

vertex (v)	dist	prev
s	0	—
a	∞	—
b	∞	—
c	∞	—
d	∞	—

$\therefore Q = \{s, a, b, c, d\}$

& visited = { }

step (2)



time Appointment Notes

step 2)

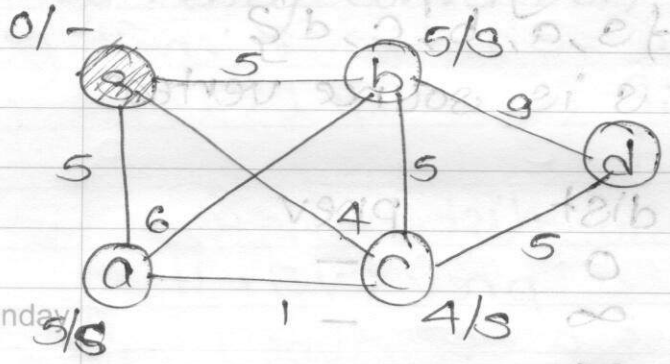
let u → min dist[u] from Q

∴ u = s

v → neighbor (unvisited) of u

v = {a, b, c}

neighbor v	dist[v]	dist[u] + ω(u,v)	updated dist[v]	prev[v]
a	∞	0+5 = 5	5	s
b	∞	0+5 = 5	5	s
c	∞	0+4 = 4	4	s



~~step 3)~~

∴ Q = {a, b, c, d}

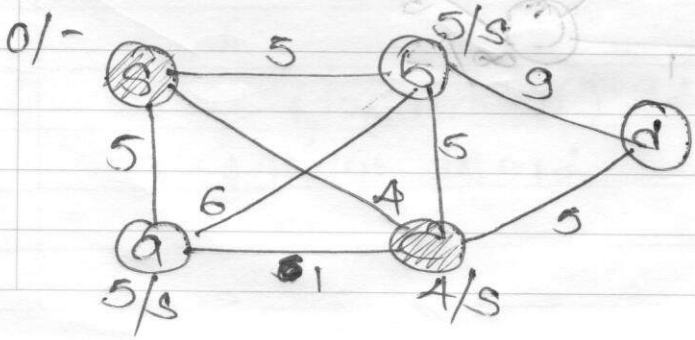
visited = {s}

step 3)

u = c

v = {a, b, d}

neighbor v	dist[v]	dist[u] + ω(u,v)	updated dist[v]	prev[v]
a	5	4+1 = 5	5	s
b	5	4+5 = 9	5	s
d	∞	4+5 = 9	9	c



Notes | Appointment time

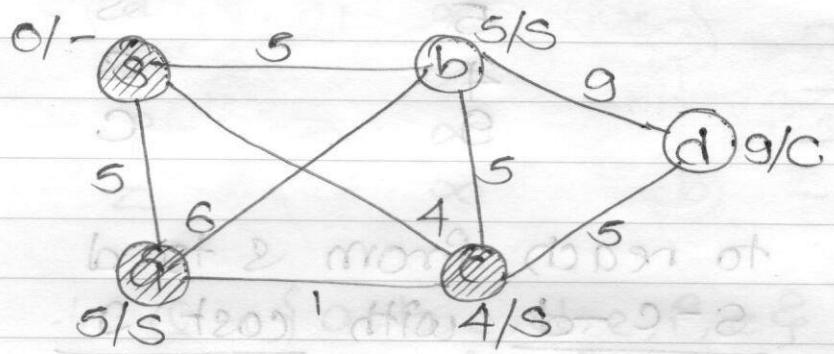
$\therefore Q = \{a, b, d\}$

visited $= \{s, c\}$

step 4) $u = a$

$v = \{b\}$

neighbor v	dist[v]	dist[u] + ω(u,v)	updated dist[v]	prev[v]
b	5	5 + 6 = 11	5	s



$\therefore Q = \{b, d\}$

visited $= \{s, c, a\}$

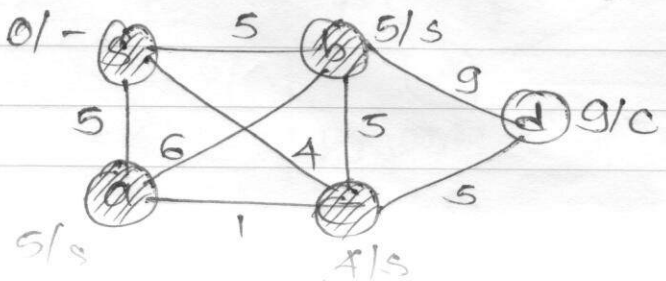
step 5) $u = b$

$v = \{d\}$

neighbor v	dist[v]	dist[u] + ω(u,v)	updated dist[v]	prev[v]
d	9	5 + 9 = 14	9	c

$\therefore Q = \{d\}$

visited $= \{s, c, a, b\}$



time Appointment Notes

Step 6) $u = d$

$v = \{\}$

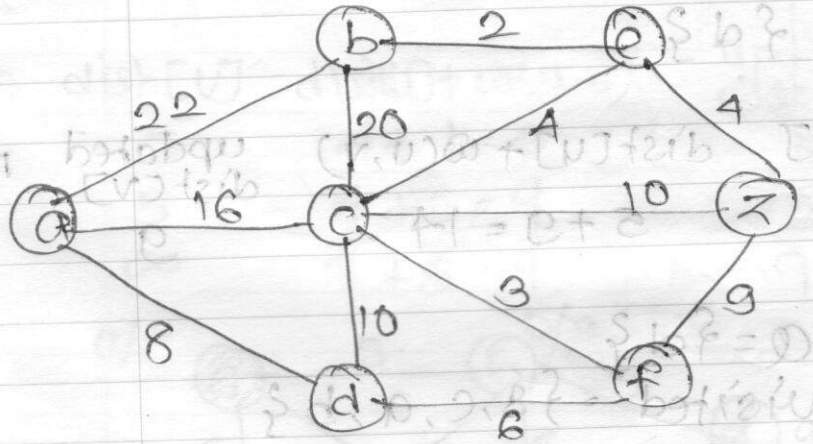
∴ finally distance from source vertices vertex s to all other is as follows

vertex (v)	dist (from s to v)	prev (v)
s	0	–
a	5	s
b	5	s
c	4	s
d	9	c

∴ to reach from s to d

path s - c - d with cost 9

ex ② Find shortest path from a to z vertex vertex using dijkstra's algorithm on following graph



Notes | Appointment | time

in above graph $G = \{V, E\}$

$V = \{a, b, c, d, e, f, z\}$

where a is source vertex

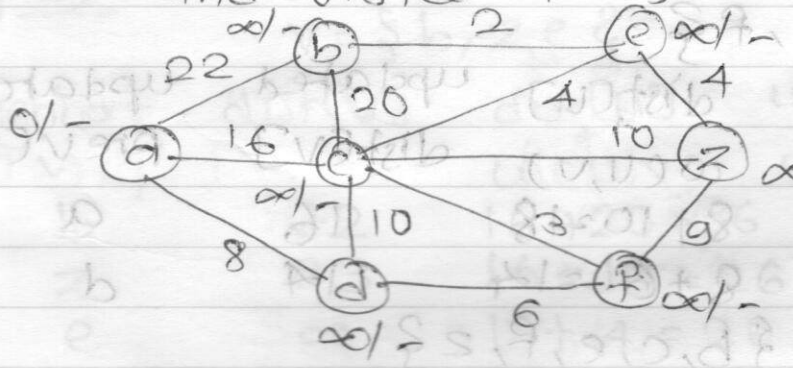
and z is destination vertex

step 1)

vertex (v)	dist (v)	prev (v)
a	0	—
b	∞	—
c	∞	—
d	∞	—
e	∞	—
f	∞	—
z	∞	—

$\therefore Q = \{a, b, c, d, e, f, z\}$

and visited = $\{\ \}$



step 2) let $u = \min(dist[U])$ from Q

$\therefore u = a$

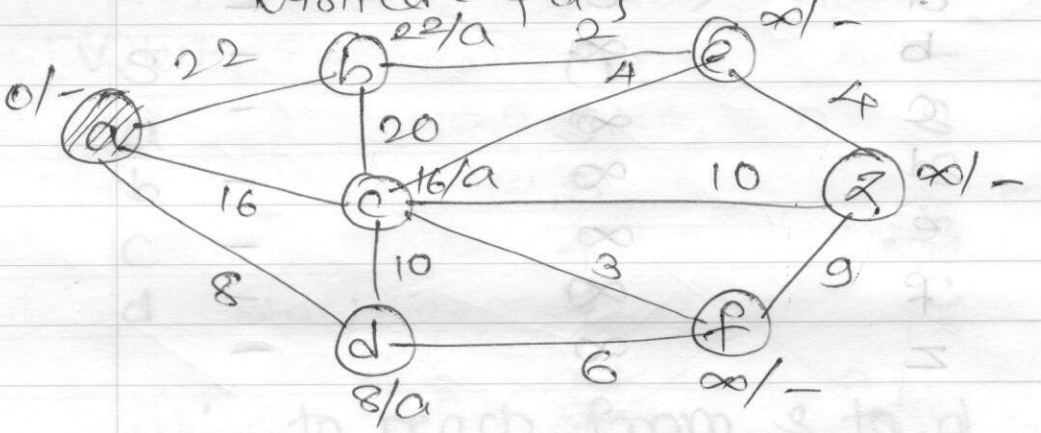
$V \rightarrow$ neighbor (unvisited) of U

$\therefore V = \{b, c, d\}$

time Appointment Notes

v neighbor	dist[v]	dist[u] + w(u,v)	updated dist[v]	updated prev[v]
b	∞	0+22=22	22	a
c	∞	0+16=16	16	a
d	∞	0+8=8	8	a

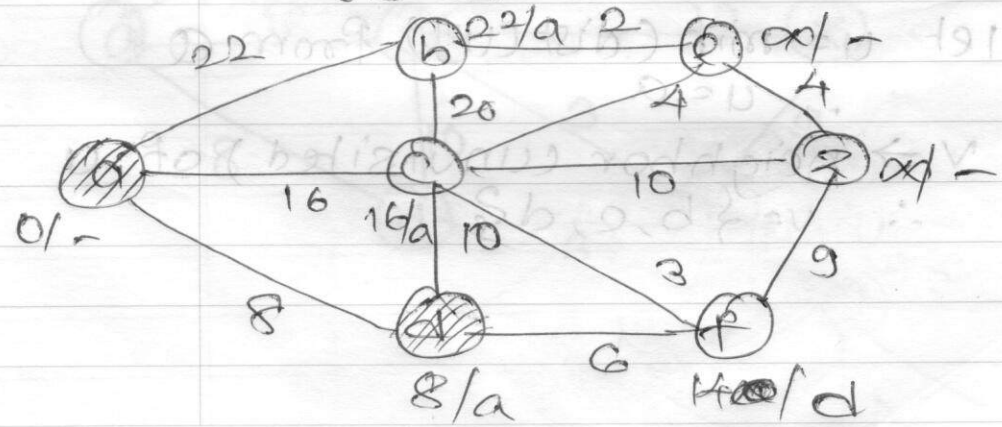
∴ Q = {b, c, d, e, f, z}
visited = {a}



step 3) u = d
v = {c, f}

v neighbor	dist[v]	dist[u] + w(u,v)	updated dist[v]	updated prev[v]
c	16	8+10=18	16	a
f	∞	8+6=14	14	d

∴ Q = {b, c, e, f, z}
visited = {a, d}



Notes | Appointment time

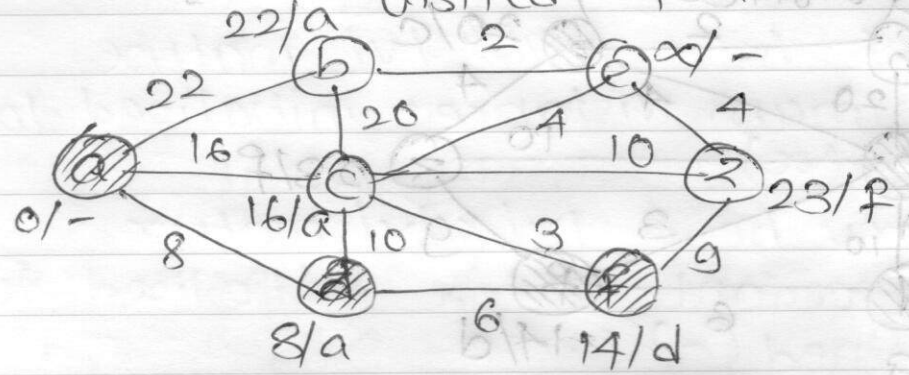
step 4) $u = f$

$v = \{c, z\}$

neighbor v	dist[v]	dist[u] + c(u,v)	updated dist[v]	updated prev[v]
c	16	14+3=17	16	a
z	∞	14+9=23	23	f

$\therefore Q = \{b, c, e, z\}$

visited = {a, d, f}



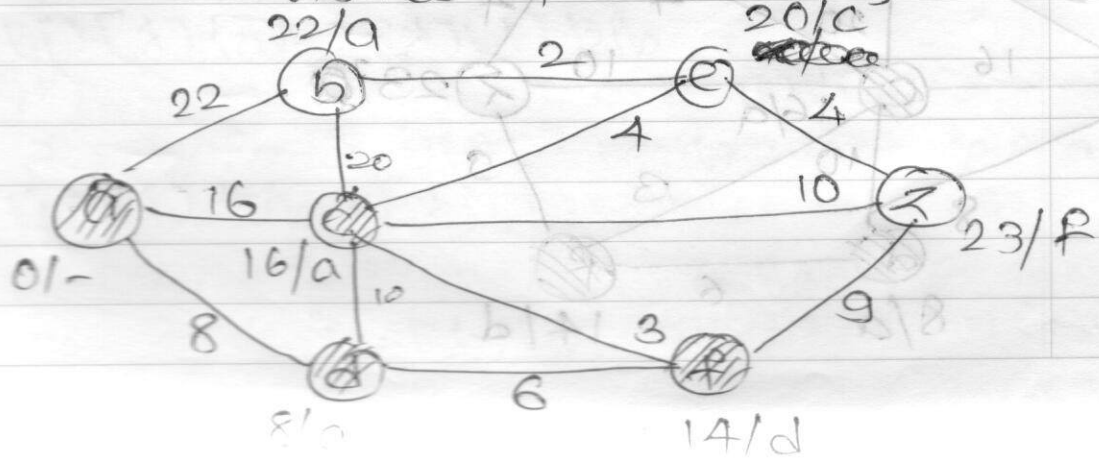
step 5) $u = c$

$v = \{b, z, e\}$

neighbor v	dist[v]	dist[u] + c(u,v)	updated dist[v]	updated prev[v]
b	22	16+20=36	22	a
z	23	16+10=26	23	f
e	∞	16+4=20	20	c

$\therefore Q = \{b, e, z\}$

visited = {a, d, f, c}



time | Appointment | Notes

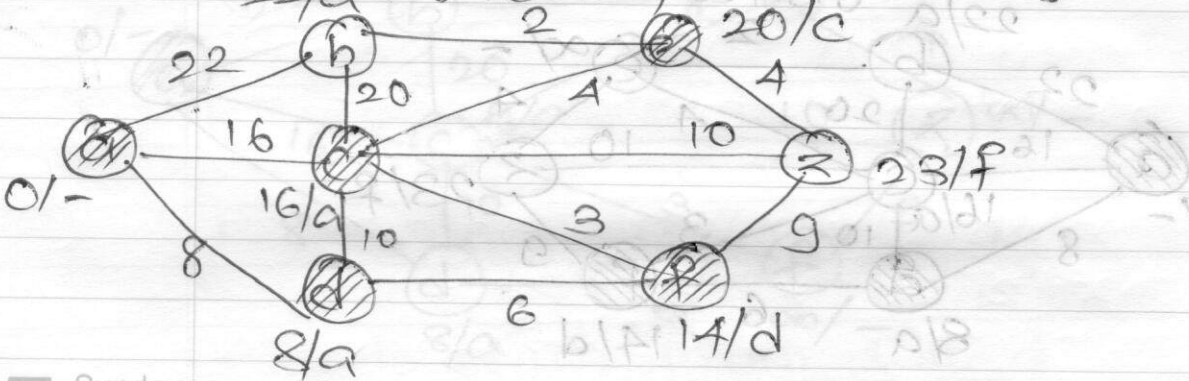
step 6) $u = e$

$v = \{b, z\}$

neighbor v	dist[v]	dist[u] + c(u,v)	updated dist[v]	updated prev[v]
b	22	20+2=22	22	a
z	23	20+4=24	23	f

$\therefore Q = \{b, z\}$

visited = {a, d, f, c, e}



25 Sunday

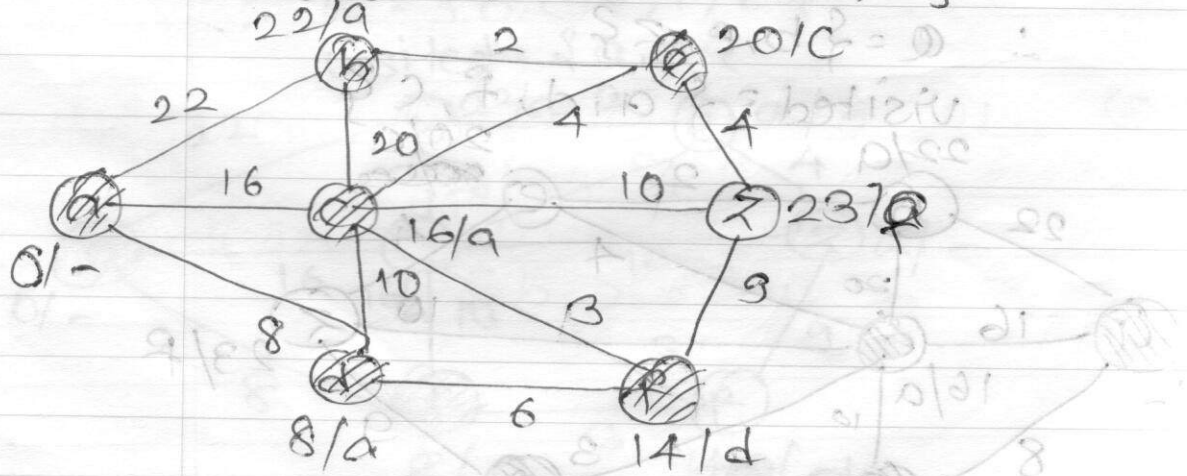
step 7) $u = b$

$v = \{z\}$

neighbor v	dist[v]	dist[u] + c(u,v)	updated dist[v]	updated prev[v]
z	23	22+2=24	23	f

$\therefore Q = \{e\}$

visited = {a, d, f, c, e, b}



Notes | Appointment | time

step 8) $u = \{z\}$
$v = \{\}$

since z is destination node we have found shortest path from a to z

path : a - d - f - z

cost : 23

time Appointment Notes

Floyd - Warshall Algorithm

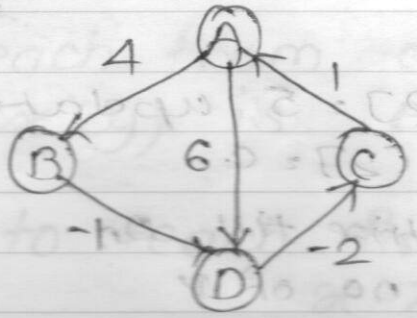
- used to find shortest path in a graph
- between any vertex to all other vertices
- graph can contain -ve edge
- graph should not have -ve edge cycle reachable from source node

Algorithm

1) let dist be a |V| x |V| matrix of minimum distances initialized to infinity (∞)
2) let prev be a |V| x |V| matrix of precedant vertex initialized to undefined (-)
3) for each vertex v in G = {V, E}
   dist[v][v] = 0
   prev[v][v] = - (undefined)
4) for each edge e(u,v) in G
   dist[u][v] = w(u,v)
   prev[u][v] = u
5) for k = 1 to v
   for i = 1 to v
   for j = 1 to v
   if (dist[i][j] > dist[i][k] + dist[k][j])
   then
   dist[i][j] = dist[i][k] + dist[k][j]
   and
   prev[i][k] = i

Notes | Appointment | time

ex (1)



$G = \{V, E\}$

$V = \{A, B, C, D\}$

step 1) initial matrix

$$D_0 = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 4 & \infty & 6 \\ \infty & 0 & \infty & -1 \\ 1 & \infty & 0 & \infty \\ \infty & \infty & -2 & 0 \end{bmatrix} \end{array} \qquad P_0 = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} - & A & - & A \\ - & - & - & B \\ C & - & - & - \\ - & - & D & - \end{bmatrix} \end{array}$$

step 2) note: copy first col & first row of D_0 & P_0 as it is in D_1 & P_1 resp.

$$D_1 = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 4 & \infty & 6 \\ \infty & 0 & \infty & -1 \\ 1 & 5 & 0 & 7 \\ \infty & \infty & -2 & 0 \end{bmatrix} \end{array} \qquad P_1 = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} - & A & - & A \\ - & - & - & B \\ C & A & - & A \\ - & - & D & - \end{bmatrix} \end{array}$$

note:

1st row, 1st col of D_1 are now pivot row, col resp.

if row/col contains ∞ then copy that row as it is from prev (i.e. 2nd, 4th row) & (3rd col) in D_1 & P_1

copy 2nd, 4th row, & 3rd col from D_0 to D_1 as it is

now to decide value of $D_1[3][2] = \min(D_0[3][2],$

$D_1[3][1] +$

$D_1[1][2])$

time | Appointment | Notes

if D[i][j] is updated then respective P[i][j] = i

∴ $D_1[3][2] = 5$ updated

∴ $P_1[3][2] = C$

continue creating matrix till D_n where n is total no of V

D_n matrix will give shortest path from any vertex to any other vertex

P_n matrix will give reachability of one vertex from another.

step 3)

$D_2 =$

	A	B	C	D
A	0	4	∞	3
B	∞	0	∞	-1
C	1	5	0	4
D	∞	∞	-2	0

$P_2 =$

	A	B	C	D
A	—	A	—	B
B	—	—	—	B
C	C	A	—	B
D	—	—	D	—

step 4)

$D_3 =$

	A	B	C	D
A	0	4	∞	3
B	∞	0	∞	-1
C	1	5	0	4
D	-1	3	-2	0

$P_3 =$

	A	B	C	D
A	—	A	—	B
B	—	—	—	B
C	C	A	—	B
D	C	C	D	—

step 5)

$D_4 =$

	A	B	C	D
A	0	4	1	3
B	-2	0	-3	-1
C	1	5	0	4
D	-1	3	-2	0

$P_4 =$

	A	B	C	D
A	—	A	D	B
B	D	—	D	B
C	C	A	—	B
D	C	C	D	—

Notes | Appointment time

note:

cost to reach from one vertex to another (xe) vertex (y) is $D_n[x][y]$

path to reach from xe to y is x-z-y

where z = null if $p_n[x][y] = x$
else $z = p_n[x][c] - c$

ex. shortest path from C to D is

cost $D_4[C][D] = 4$

path C - z - D

$= C - (P_4[C][D]) - D$

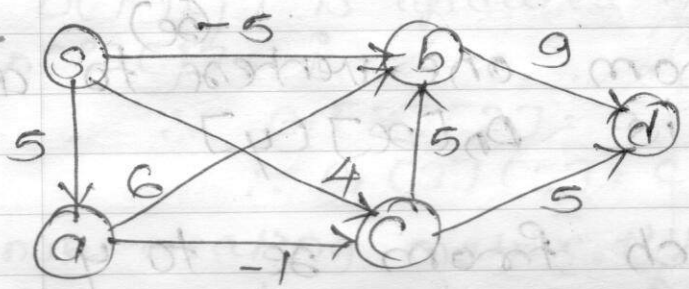
$= C - (P_4[C][B]) - B - D$

$= C - (P_4[C][A]) - A - B - D$

$= C - A - B - D$

time | Appointment | Notes

ex (2)



find shortest path from s to d

step 1)

$D_0 =$

	s	a	b	c	d
s	0	5	-5	4	∞
a	∞	0	6	-1	∞
b	∞	∞	0	∞	9
c	∞	∞	5	0	5
d	∞	∞	∞	∞	0

$P_0 =$

	s	a	b	c	d
s	—	S	S	S	—
a	—	—	A	A	—
b	—	—	—	—	B
c	—	—	C	—	C
d	—	—	—	—	—

step 2)

1 Sunday January

$D_1 =$

	s	a	b	c	d
s	0	5	-5	4	∞
a	∞	0	6	-1	∞
b	∞	∞	0	∞	9
c	∞	∞	5	0	5
d	∞	∞	∞	∞	0

$P_1 =$

	s	a	b	c	d
s	—	S	S	S	—
a	—	—	A	A	—
b	—	—	—	—	B
c	—	—	C	—	C
d	—	—	—	—	—

step 3)

$D_2 =$

	s	a	b	c	d
s	0	5	-5	4	∞
a	∞	0	6	-1	∞
b	∞	∞	0	∞	9
c	∞	∞	5	0	5
d	∞	∞	∞	∞	0

$P_2 =$

	s	a	b	c	d
s	—	S	S	S	—
a	—	—	A	A	—
b	—	—	—	—	B
c	—	—	C	—	C
d	—	—	—	—	—

Notes | Appointment time

step 4)

D_3 =

	s	a	b	c	d
s	0	5	-5	4	4
a	∞	0	6	-1	15
b	∞	∞	0	∞	9
c	∞	∞	5	0	5
d	∞	∞	∞	∞	0

P_3 =

	s	a	b	c	d
s	—	S	S	S	B
a	—	—	A	A	B
b	—	—	—	—	B
c	—	—	C	—	C
d	—	—	—	—	—

step 5)

D_4 =

	s	a	b	c	d
s	0	5	-5	4	4
a	∞	0	4	-1	4
b	∞	∞	0	∞	9
c	∞	∞	5	0	5
d	∞	∞	∞	∞	0

P_4 =

	s	a	b	c	d
s	—	S	S	S	B
a	—	—	C	A	C
b	—	—	—	—	B
c	—	—	C	—	C
d	—	—	—	—	—

step 6)

D_5 =

	s	a	b	c	d
s	0	5	-5	4	4
a	∞	0	4	-1	4
b	∞	∞	0	∞	9
c	∞	∞	5	0	5
d	∞	∞	∞	∞	0

P_5 =

	s	a	b	c	d
s	—	S	S	S	B
a	—	—	C	A	C
b	—	—	—	—	B
c	—	—	C	—	C
d	—	—	—	—	—

to reach from s to d

cost $D_5[s][d] = 4$

path = $s - (P_5[s][d]) - d$

$= s - (P_5[s][b]) - b - d$

$= s - b - d$

here $D_5[a][s] = \infty$ means

sa is not reachable from a

by any path

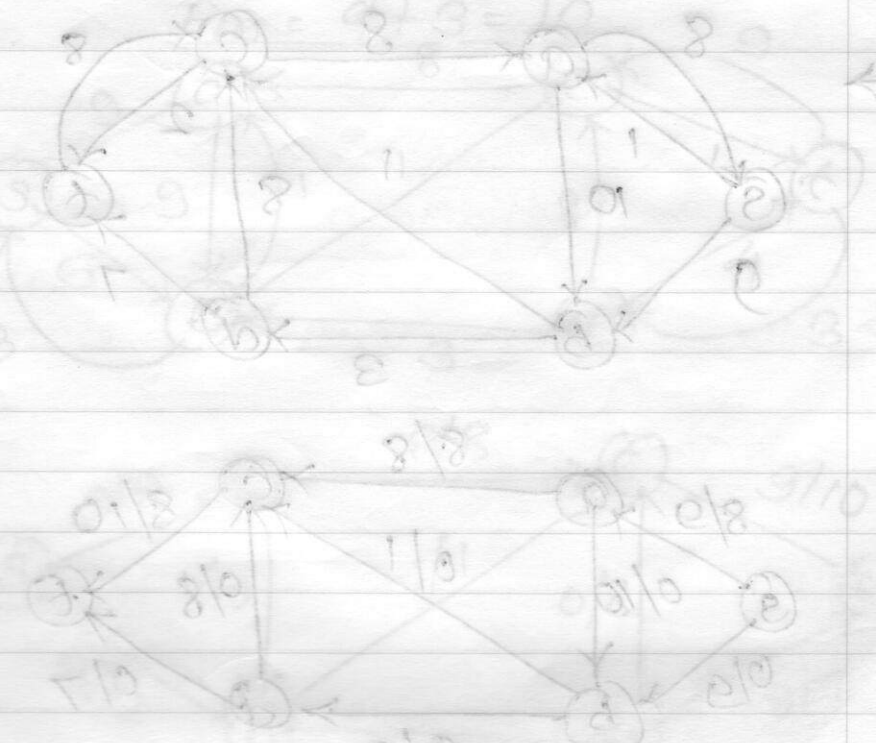
time Appointment Notes

Network flow problem

- also called as max flow problem
- solved using ford-Fulkerson algorithm
- it tries to find maximum flow in a network from a single source s to a single sink t
- practical applications
  1. freight - find the maximum no of trucks we can send out to a location according to road map with different road capacities.
  2. networking - no of packets that can be send from one router to another via intermediate routers as per routing table
  3. maximum water that can be sent via pipelines to a city
- its requirements.
  simple direct graph
  1 source, 1 sink vertex
  no -ve capacity for any edge (such edge will be treated as capacity 0)
- Residual graph (G_f)
  - only edges from G that can still have more flow capacity.
    $c_f(u,v) = c(u,v) - f(u,v)$
  - add edges (reverse direction) to decrease flow
    $c_f(v,u) = c_f(v,u) + f(u,v)$

- Augmenting paths (p)
path from s to t in G_f
residual capacity = smallest capacity of edges of P

Algorithm
1) $f_m = 0$
2) while there exist augmenting path P in G_f
   a) $c_f(p)$ = smallest capacity on P
   b) $f_m = f_m + c_f(p)$
   c) for each edge on P
      $c_f(u,v) = c_f(u,v) - c_f(p)$
      $c_f(v,u) = c_f(v,u) + c_f(p)$
      (add reverse edge if not present)



time Appointment Notes

example (1)

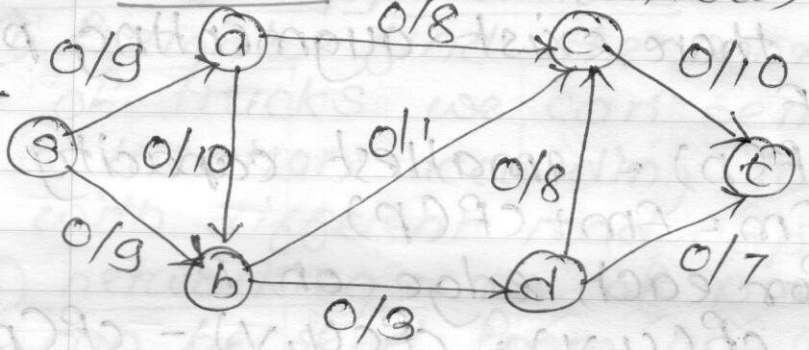
from given graph

step 1)

$F(u,v) = 0$ For all edges

$Fm = 0$ (max flow)

flow network (FN) →



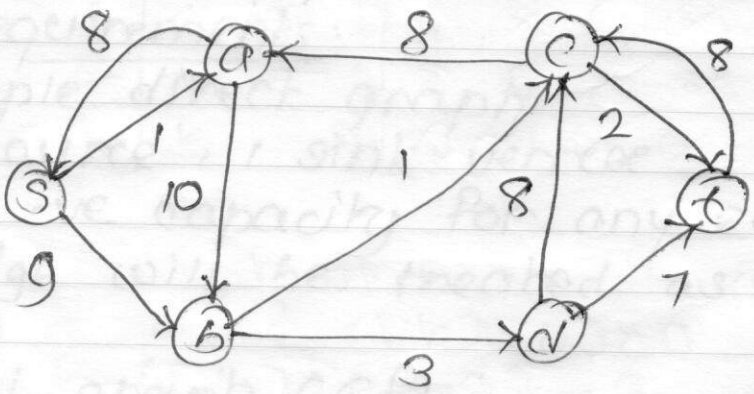
step 2) augmenting path (P) = s - a - c - t

$CF(P) = 8$

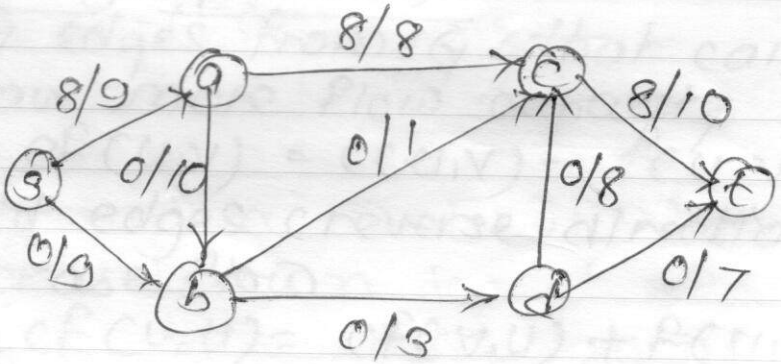
$\therefore Fm = 0 + 8 = 8$

8 Sunday

Residual Graph → (GF)



modified FN

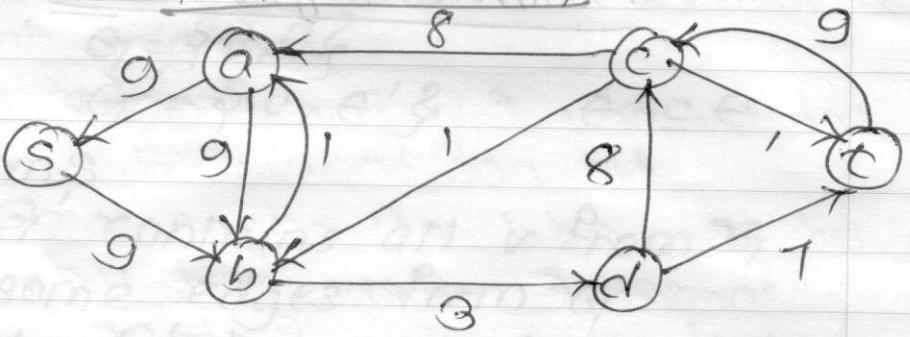


step 3) $P = S - a - b - c - t$

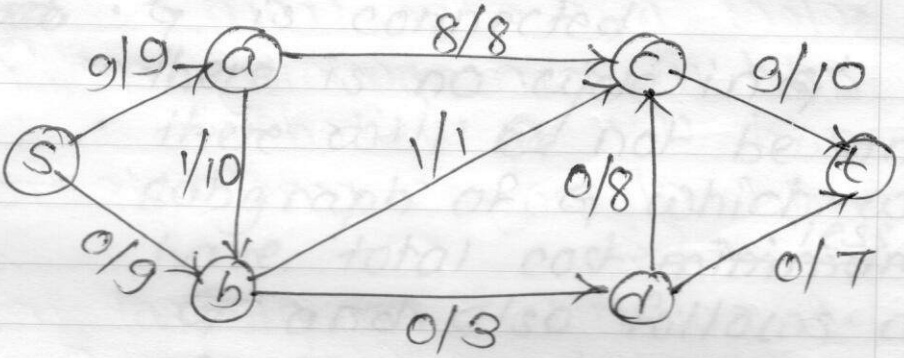
$C_f(P) = 01$

$Fm = 8 + 1 = 09$

GF



FN

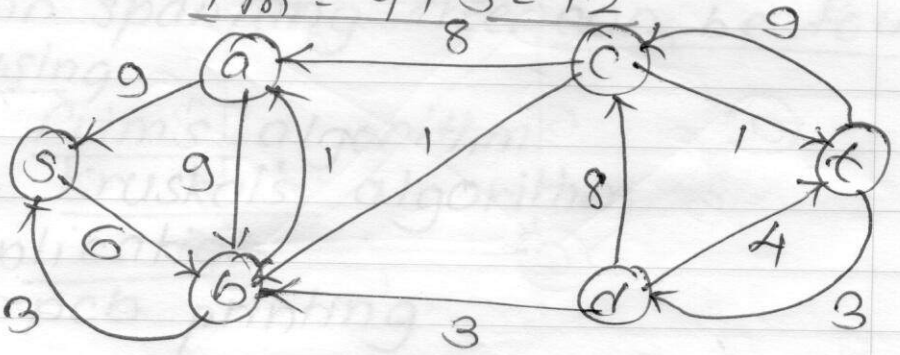


step 4) $P = S - b - d - t$

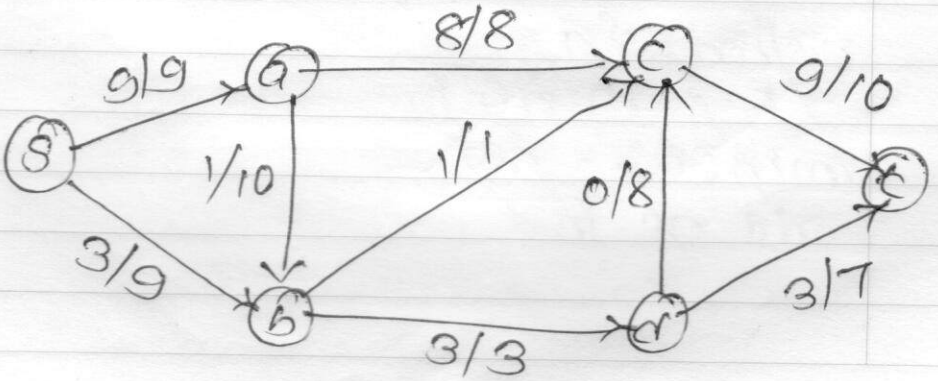
$C_f(P) = 3$

$Fm = 9 + 3 = 12$

GF

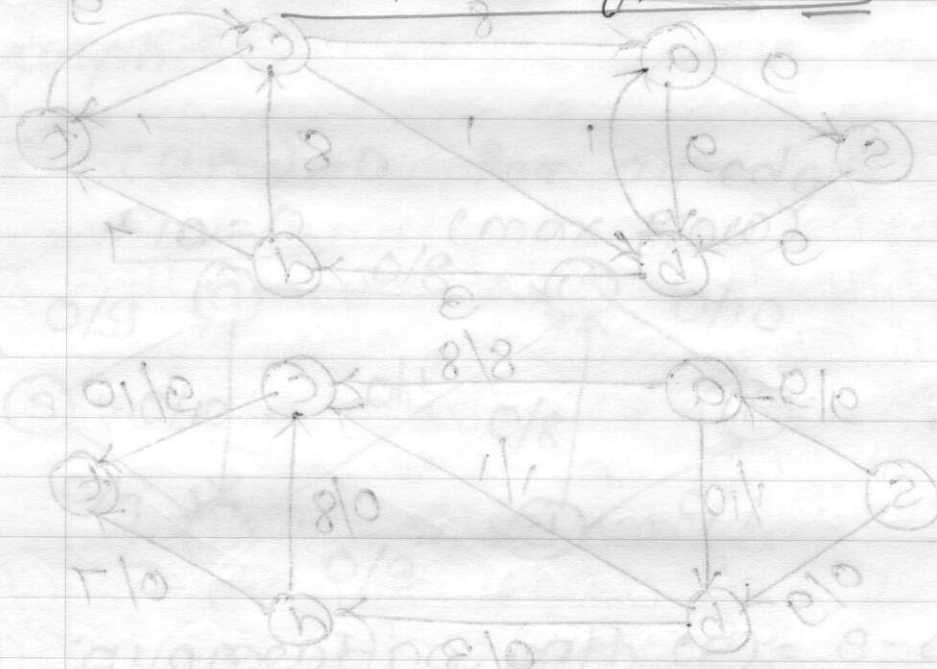


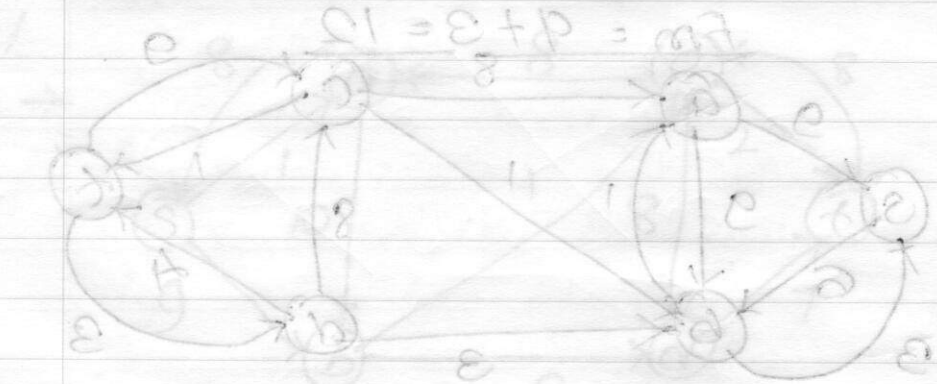
FN

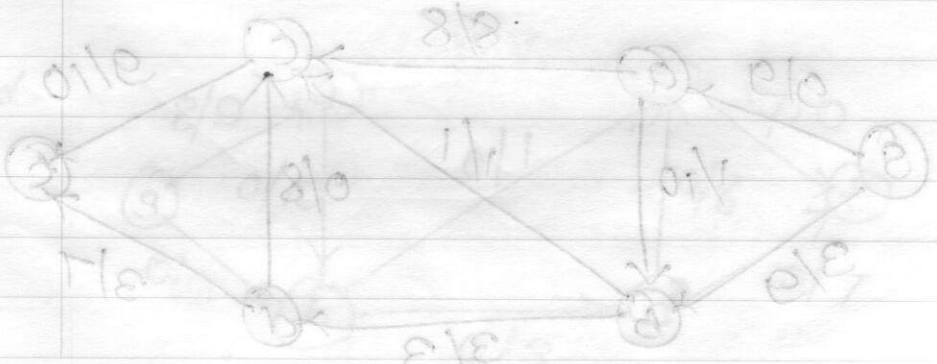


time | Appointment | Notes

step 5) no augmenting path
∴ from s to t
maximum flow = 12







Notes | Appointment | time

minimum spanning tree

- finding a subgraph G' of graph G such that
  $G = \{V, E\}$
  $G' = \{v, E'\}$ $E' \subseteq E$

  means
  - 'G' contains all v from G
  - some edges from G
  - G' is connected
  - there is no cycle in G'
  - there will not be another subgraph of G which will have total cost less than G' and also follows all necessary properties of G'
- such a G' is called as minimum spanning tree for G
- min spanning tree can be found using
  Prim's algorithm
  or Kruskal's algorithm
- application
  - pcb printing

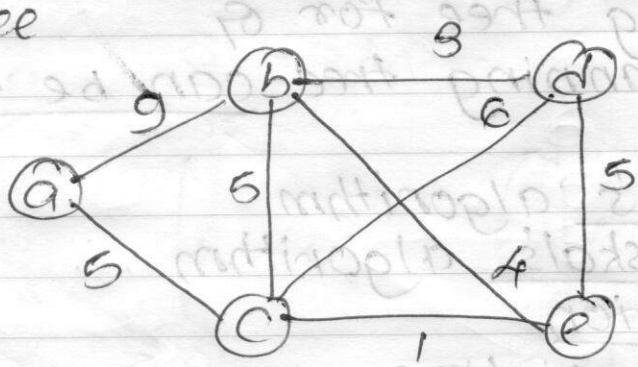
time | Appointment | Notes

Prim's algorithm

1) Read a graph G = {V, E}
2) visited = { } // vertices on min spanning tree
3) let u = any vertex from V but not in visited
4) add all edges (u, v) in set E where v ∉ visited
5) select min edge from E
6) add in min spanning tree, remove from E
7) add u, v in visited }
8) add all edges reachable from v in E }
9) if not all v are present in visited } then continue from step 5
10) else show minimum spanning tree

ex.

G =



step 1)

let u = a
visited = { a }
E = { ab, ac }
min(E) = ac
add ac in

Notes | Appointment | time

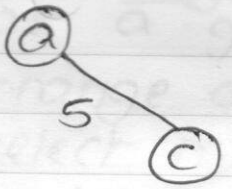
Step 1) let u = a

visited = {a}

E = edges (u,v)	w(u,v)
ab	9
ac	5

min(E) = ac

add ac in min spanning tree



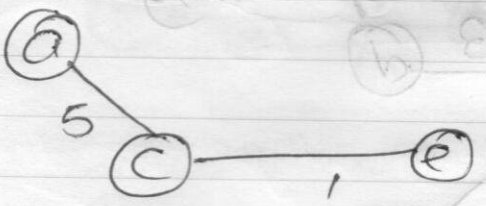
Step 2) visited = {a, c}

E =

e(u,v)	w(u,v)
ab	9
bc	5
ce	1
cd	6

min(E) = ce

add ce in min spanning tree



time | Appointment | Notes

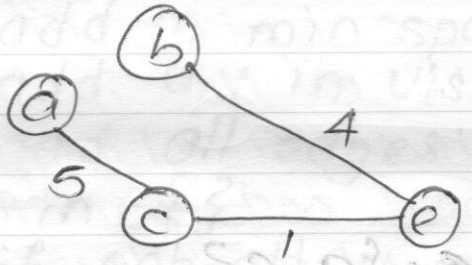
step 3) visited = {a, c, e}

E =

e(u,v)	w(u,v)
ab	9
bc	5
cd	6
be	4
de	5

min(E) = be

add be in min spanning tree



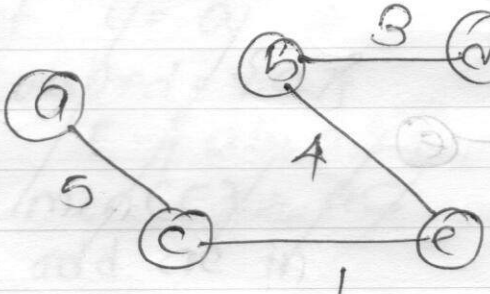
step 4) visited = {a, c, e, b}

E =

e(u,v)	w(u,v)
ab	9
bc	5
cd	6
de	5
bd	3

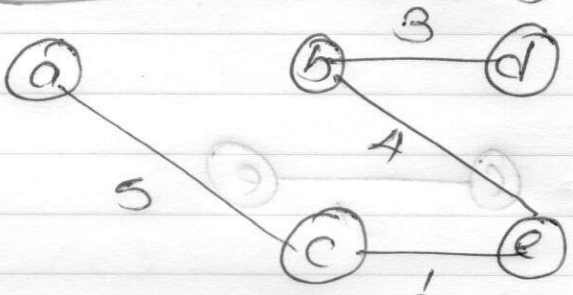
min(E) = bd

add bd in min spanning tree



∴ visited = {a, b, c, d, e}

∴ minimum spanning tree is

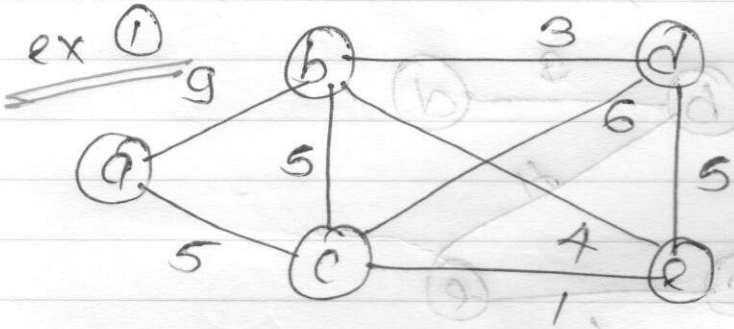


total cost = 13

kruskal's algorithm

1) read a graph $G = \{V, E\}$
2) arrange all edges in ascending order
3) select min edge e
   add in min spanning tree
   if it is not creating cycle
   else discard
4) ~~did~~ ~~as~~ ~~there~~ continue step 3 for all edges
5) display min spanning tree

ex (1)



	e(u,v)	w(u,v)
1)	ce	1
2)	bd	3
3)	be	4
4)	ac	5
5)	bc	5
6)	de	5
7)	cd	6
8)	ab	9

time Appointment Notes

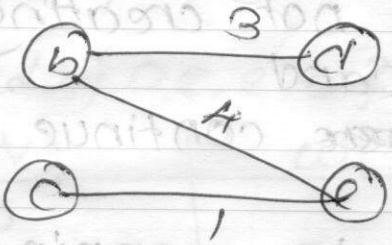
step 1) add ce in min spanning tree (m)

m =

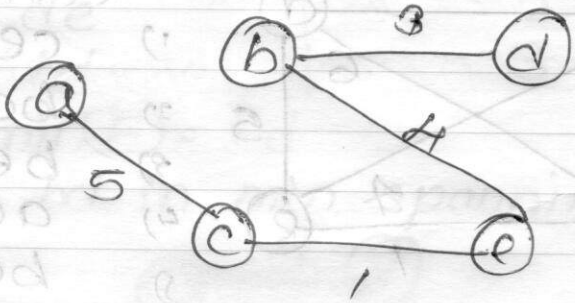
step 2) add bd in m

m =

Step 3) add be in m



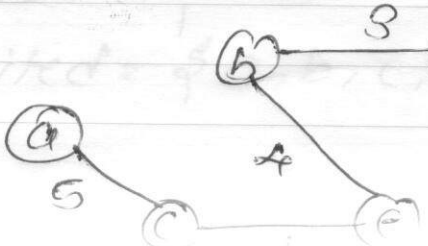
step 4) add ac in m



step 5) discard all other edges as they create cycle in m

step 6)

minimum spanning tree ⇒



total cost = 13