✱ Graph

- Consist of no.of nodes & Linkes ..

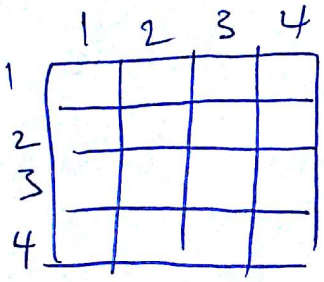
two type

1- directed

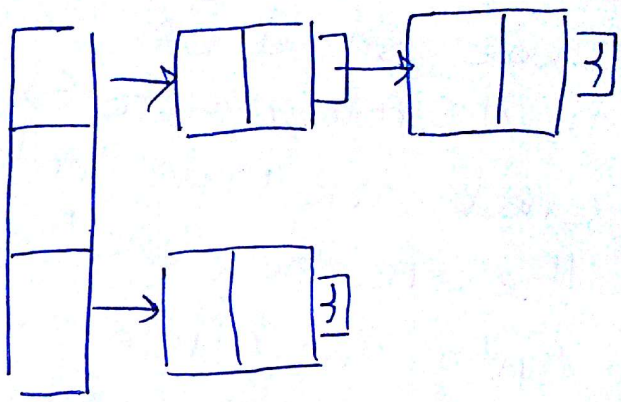
2- Undirected "symmetric around it's dragonal"

→ Graph Can be represent

using array



using linked list



✱ All-to-All shortest path

- select path with min weight between All Nodes

Steps

1- Create original array from Graph
2- select source & destination
3- Choose path with min weight
4- change destination
5- Repeat 3 & 4 till same source reach All destinations.
6- Choose another source & repeat + update original array.

Shortest path :-

select path with min cost between src & destination

Steps

1- Create original array from Graph

2- Create an extra matrix
- no of nodes = no. of rows
- no of column = 3

From cost bol

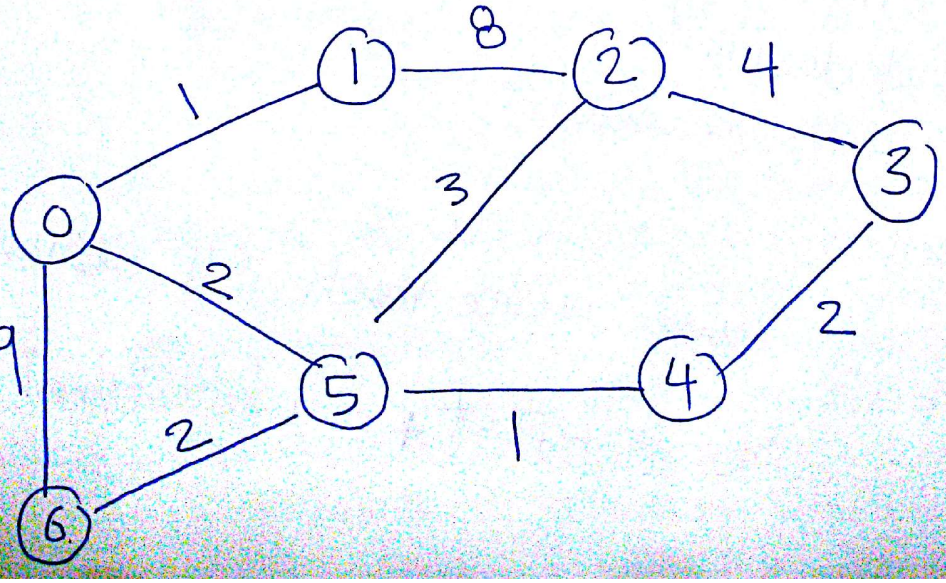
-1	∞	F
-1	∞	F
-1	∞	F

3- choose src & des then put src in a circle, update extra matrix

4- check the Node that directly connected to the src then select one with min cost to Enter the circle

5- repeat 4 with the New nodes till reach our des..

Ex Find shortest path between 0 & 6



Solution

1- Create weight matrix from Graph

	0	1	2	3	4	5	6
0	0	1	#	#	#	~~4~~	9
1	1	0	8	#	#	#	#
2	#	8	0	4	#	3	#
3	#	#	4	0	2	#	#
4	#	#	#	2	0	1	#
5	2	#	3	#	1	0	2
6	9	#	#	#	#	2	0

2- Create Extra matrix

	From	Cost	bol
0	-1	∞	F
1	-1	∞	F
2	-1	∞	F
3	-1	∞	F
4	-1	∞	F
5	-1	∞	F
6	-1	∞	F

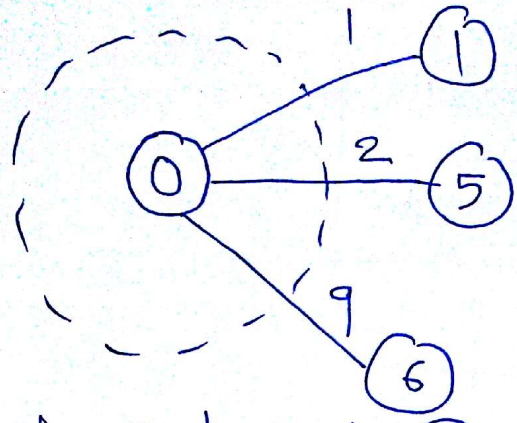
3- Choose src & des, then update Extra matrix

source : 0

des : 6

0	0	0	T
1	-1	∞	F
2	-1	∞	F
3	-1	∞	F
4	-1	∞	F
5	-1	∞	F
6	-1	∞	F

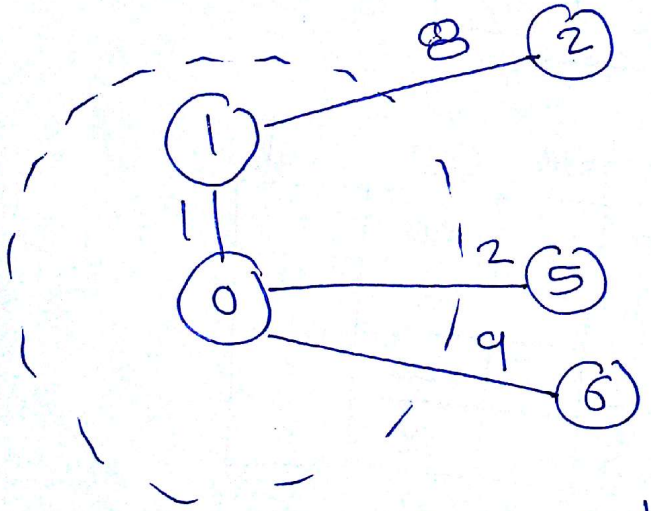
4- Put src inside circle & check connected Node for min Cost.



0	0	0	T
1	~~5~~	~~∞~~	F
2	-1	∞	F
3	-1	∞	F
4	-1	∞	F
5	~~0~~	~~2~~	F
6	~~9~~	~~9~~	F

∴ min Cost Node (1), Cost = 1
∴ Enter circle & update extra Matrix

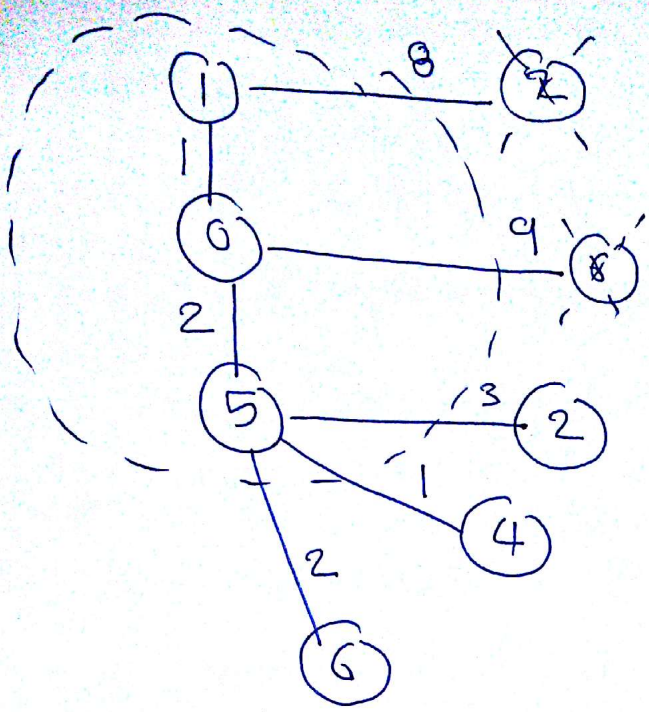
5-



0	0	0	T
1	0	1	T
2	1	~~9~~	f
3	-1	∞	f
4	-1	∞	f
5	~~0~~	~~2~~	f
6	~~0~~	~~9~~	F

∴ min Cost Node (5), Cost = 2
∴ Enter circle & update Matrix

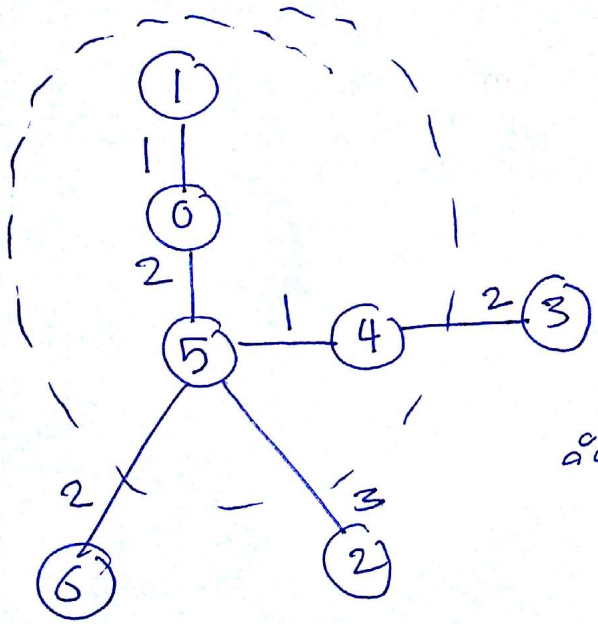
6-



0	0	0	T
1	0	1	T
2	5	5	F
3	-1	∞	F
4	5	3	F
5	0	2	T
6	5	4	F

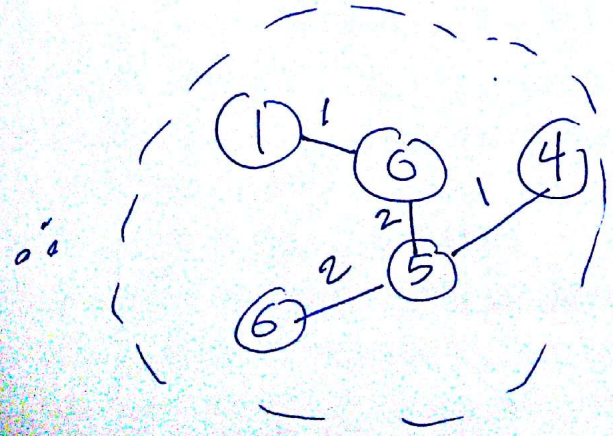
∴ min Node 4, Cost = 3

7-



0	0	0	T
1	0	1	T
2	5	5	F
3	4	5	F
4	5	3	T
5	0	2	T
6	5	4	F

∴ min Node = 6, Cost 4



∴

&

Path

⊛ Minimum spanning tree :-

The ided is to make only the path with minimum weight between all Nodes ..

steps

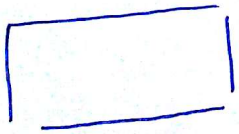
1- create original matrix from Graph

2- Create New matrix will all values = inF to represent the New Graph with All Node haven't been connected yet

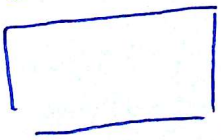
3- Create initial array to indicate node position if -1 then it's outside if else it's inside..

4- search with the two Node with the least Cost in the half of the original matrix (c<r) as it's symmetric..

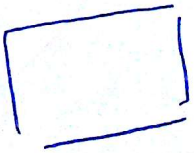
row min



Column min



min

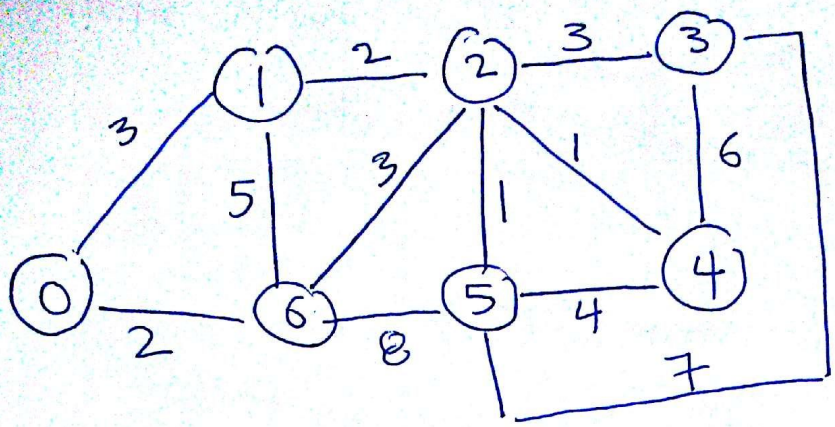


ok

5- Update initial array & the New matrix with the position of the two Nodes and the cost of them, Then delete Cost from original matrix

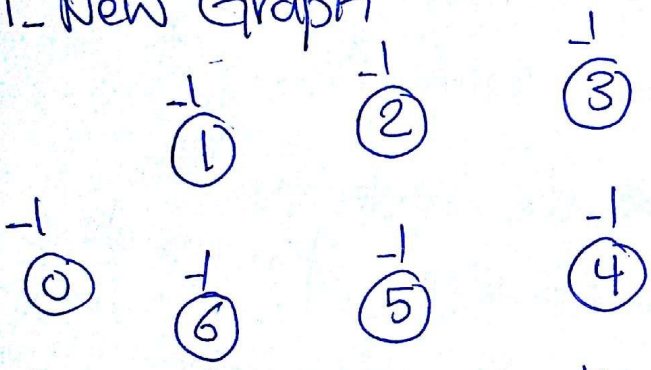
6- repeat till Cost = inf ..

Ex Find min spanning tree.

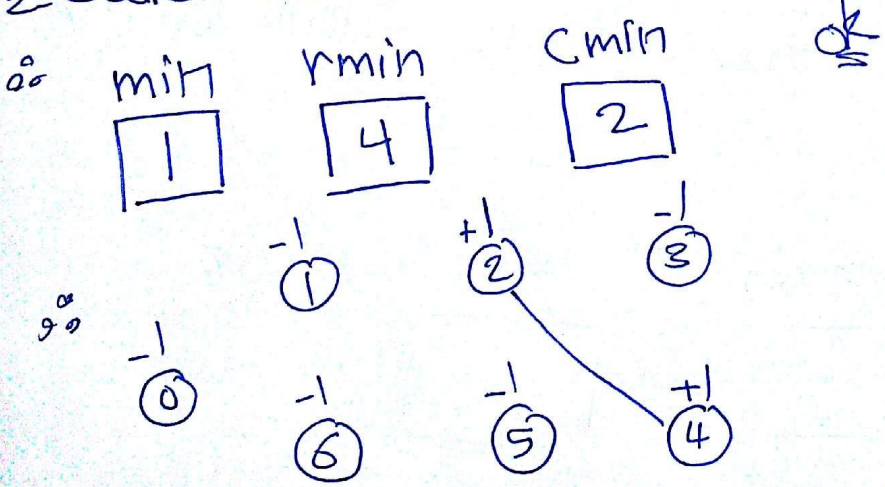


	0	1	2	3	4	5	6
0	0	3	#	#	#	#	2
1	3	0	2	#	#	#	5
2	#	2	0	3	1	1	3
3	#	#	3	0	6	7	#
4	#	#	1	6	0	4	#
5	#	#	1	7	4	0	8
6	2	5	3	#	#	8	0

1- New Graph



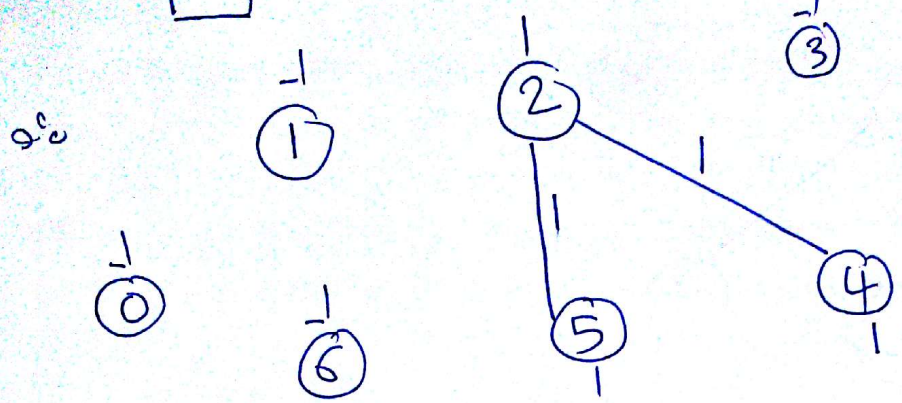
2- Search two Node with min Cost



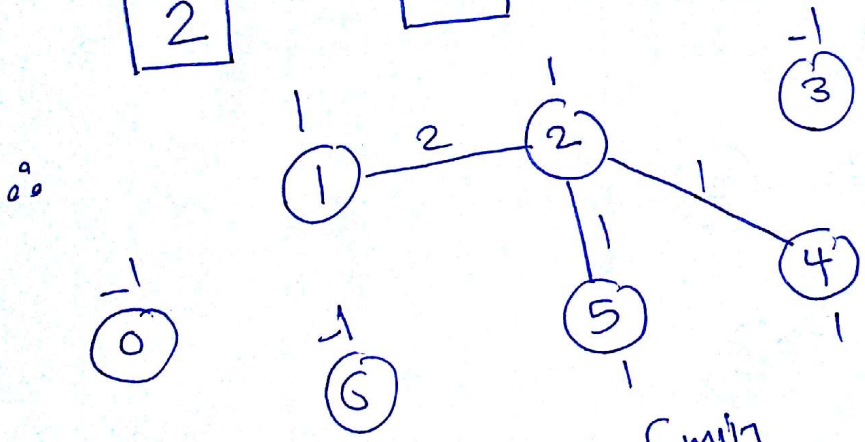
3- delete min from orginal then repeat 2

(8)

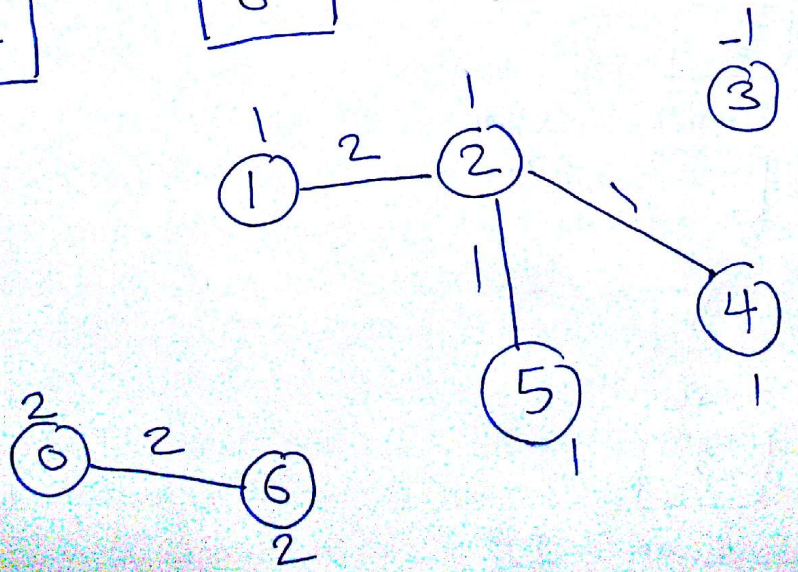
4- min [1] rmin [5] Cmin [2] ok



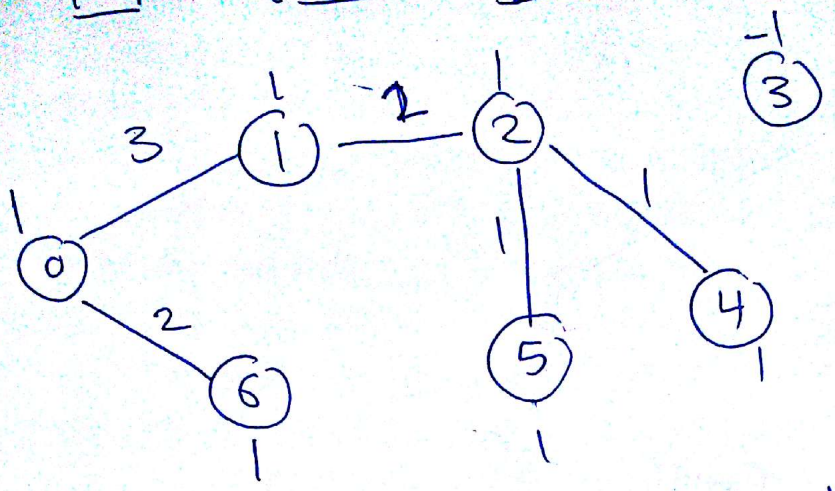
5- min [2] rmin [2] Cmin [1] ok



6- min [2] rmin [6] Cmin [0] ok

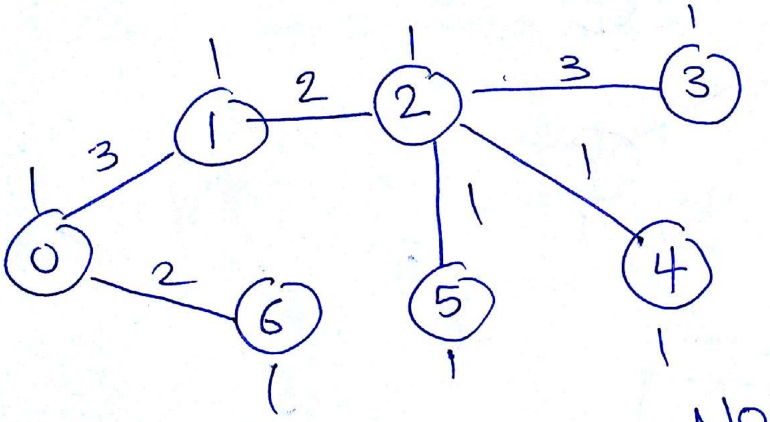


7- min = 3, rmin = 1, Cmin = 0



ok

8- min = 3, rmin = 3, Cmin = 2



No

9- min = 4, rmin = 5, Cmin = 4

No

10- min = 5, rmin = 6, Cmin = 1

No

11- min = 6, rmin = 4, Cmin = 3

No

12- min = 7, rmin = 5, Cmin = 3

No

13- min = 8, rmin = 6, Cmin = 5

Tree 3-

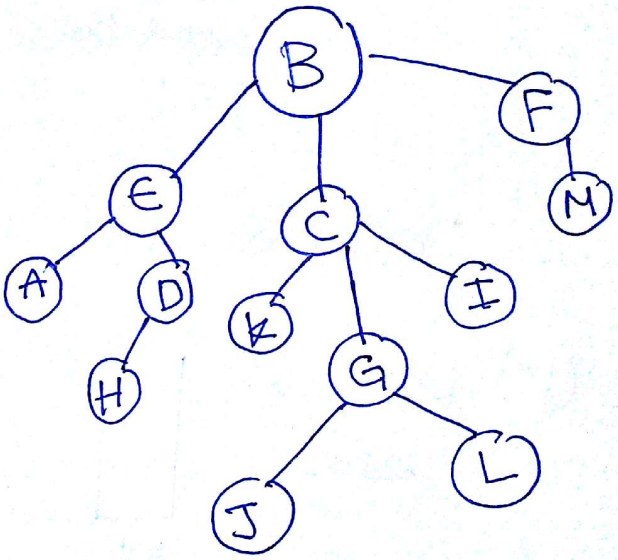
- We can represent our tree as an array of structure with three Element "Name, father, order"

Ex

Name	father	order
A	E	1
B	0	0
C	B	2
D	E	2
E	B	1
F	B	3
G	C	2
H	D	1
I	C	3
J	G	1
K	C	1
L	G	2
M	F	1

To Draw from table

1- search for root
2- search for son's



⇒ How to visit tree ?!

(4) Depth first search " based on stack concept"

Sequence

B → E → A → D → H → C → K → G → J → L → I → F → M

Implementation steps

(1) push root | B |

(2) Pop root then push it's sons from right to left

| E |
| C |
| F |

B

(3) Pop father then push it's sons

| A |
| D |
| C |
| F |

E

---- and so on until

(16) pop last son

| |

M

(2) Breadth first search - based on queue Concept

Sequence

B → E → C → F → A → D → K → G → I → M → H → J → L

Implementation steps

(1) Enqueue root

B		

(2) dequeue root then Enqueue its sons from ~~left to right~~ left to right

E	C	F

B

(3) dequeue father then Enqueue its sons.

C	F	A	D

E

:

:

:

and so on until

(14) dequeue last son

L

⇒ How to know it's Binary tree
Every Node have only 2 or 1 son..

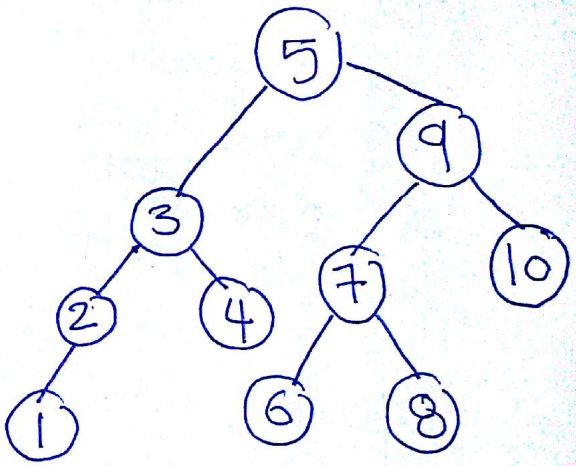
Binary search tree..

Condition : left son smaller than father
right son bigger than father

⇒ Given sequence of number
How to make Binary search tree?

5, 9, 3, 7, 6, 8, 2, 1, 10, 4

- Compare every no with root then put left or right depend on the comparison, if there is a father then also compare with it & put it left or right depend on comparison



⇒ search complexity
- Best case = log n
- worst case = n
- average = log n ..

⇒ visit search binary tree

(1) BFS 5, 3, 9, 2, 4, 7, 10, 1, 6, 8

(2) DFS 3type

(a) pre order : "FLR"

5, 3, 2, 1, 4, 9, 7, 6, 8, 10

(b) In order : "LfR" (order sequence)

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

(c) post order : "LRf"

1, 2, 4, 3, 6, 8, 7, 10, 9, 5

⇒ Delete Node

(1) if leafe ∴ delete

(2) if Node with one son
∴ delete & son take it's place -

(3) if Node with two sons
∴ delete & it's smallest right son
or it's largest Left son take it's
place ..