

ate
2/10/2017

"Perfect Hashing" :-

0	hf	→	c	a	b						
1											
2											
3											
4											
5	hf	→	d	e							
6											
7											
8											
9	hf	→			f	g					

$$h(a) = 0 \quad h_g(a) = 3$$

$$h(b) = 0 \quad h_o(b) = 5$$

$$h(c) = 0 \quad h_o(c) = 1$$

$$h(d) = 5 \quad h_s(d) = 0$$

$$h(e) = 5 \quad h_s(e) = 2$$

$$h(f) = 9 \quad h_g(f) = 4$$

$$h(g) = 9 \quad h_g(g) = 5$$

:- # of Hash functions use " $m+1$ " in ~~double~~ perfect

Hashing as there is a Hash function in every slot of Hash table.

:- # of Hash Tables are also " $m+1$ ",

m slots $\rightarrow m$ tables (HT) & 1 main Hash Table

Note :- ① The Expected No. of Probes in an unsuccessful Search of open Addressing Technique:

$$= \frac{1}{1-\alpha}$$

②

Successful Search :-

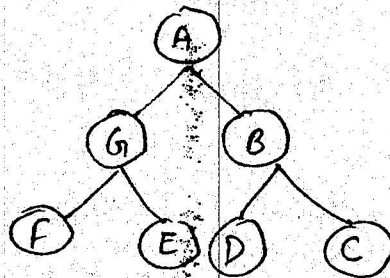
$$\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$$

"Tree & graph Traversal" :-

"Tree Traversal" :-

1. Pre-order (Root, LST, RST)
2. Post-order (LST, RST, Root)
3. In-order (LST, Root, RST)

Ex : ① :-



Pre-order :- Root then Left subtree then Right sub-tree

Pre-order :- $\begin{matrix} \text{Root} & \text{LST} & \text{RST} \\ A & G F E & B D C \end{matrix}$

In-order :- $\begin{matrix} & \text{LST} & \text{Root} & \text{RST} \\ F & G & E & A & D & B & C \end{matrix}$

Post-order :- $\begin{matrix} & \text{LST} & \text{RST} & \text{Root} \\ F & E & G & D & C & B & A \end{matrix}$

Pre-order (root) $\Rightarrow T(n)$

Post-order (root)

$\left\{ \begin{array}{l} Pf(\text{root} \rightarrow \text{Data}) \Rightarrow O(1) \\ \text{Preorder}(\text{root} \rightarrow \text{LST}) \Rightarrow T(n/2) \\ \text{Preorder}(\text{root} \rightarrow \text{RST}) \Rightarrow T(n/2) \end{array} \right\}$

$\left\{ \begin{array}{l} \text{Post-order}(\text{root} \rightarrow \text{LST}) \\ \text{Post-order}(\text{root} \rightarrow \text{RST}) \\ Pf(\text{root} \rightarrow \text{Data}) \end{array} \right\}$

Best case :-

$$T(n) = 2T(n/2) + C \quad [\text{Balanced Tree}]$$

By master's theorem

$$= O(n)$$

worst case :-

$$T(n) = C + T(n-1) + T(0) \quad [\text{unbalanced Tree}]$$

$$= T(n-1) + C$$

$$\Rightarrow O(n^2)$$

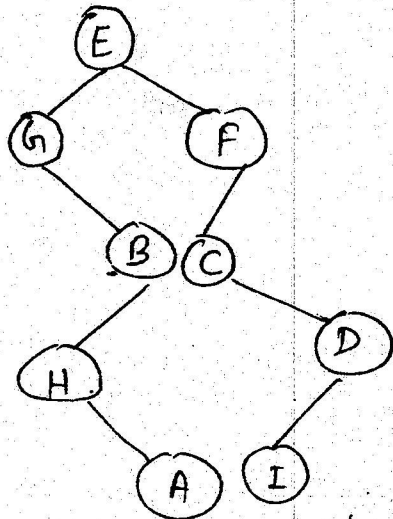
In-order (root)

$\left\{ \begin{array}{l} \text{In-order}(\text{root} \rightarrow \text{LST}) \\ Pf(\text{Root} \rightarrow \text{data}) \\ \text{In-order}(\text{root} \rightarrow \text{RST}) \end{array} \right\}$

Stack Space :- $O(\log n)$ - Best case
 $O(n)$ - worst case

Time complexity - $O(n)$ Every case } for all
 (Tree Traversal) Pre, Post, In order Traversal

Ex: ② :



In-order :

Pre-order : E G B H A F C D I

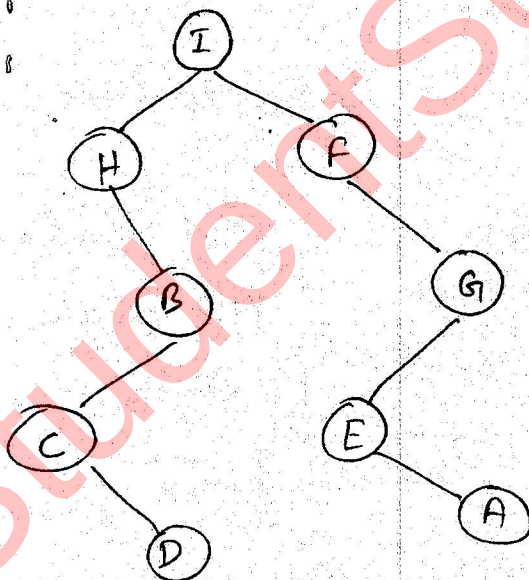
Post-order :- ~~A H~~

A H B G I D C F E

In-order :-

G B H A B E C I D F

Ex: ③ :-



In-order :-

H C D B I F E A G

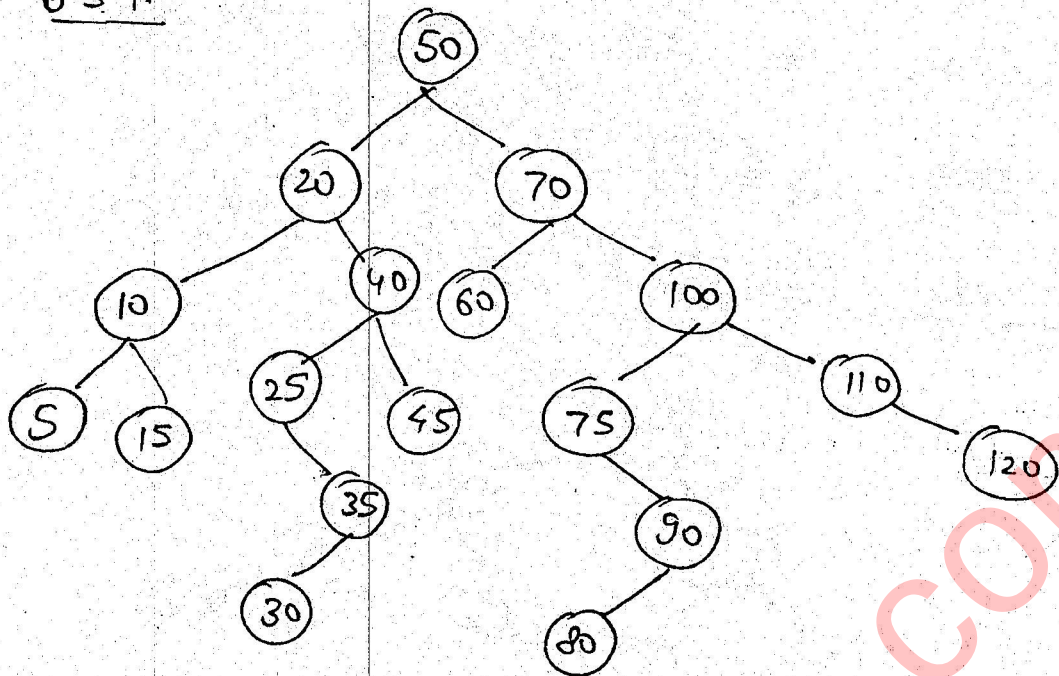
Pre-order :-

I H B C D F G E A

Post-order :-

D C B H A E G F I

Ex: (4) B.S.T.



In-order :- 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 75, 80, 90, 100, 110, 120

Pre-order :- 50, 20, 10, 5, 15, 40, 25, 35, 30, 45, 70, 60, 100, 75, 90, 80, 110, 120

Post-order :- 5, 15, 10, 30, 35, 25, 45, 40, 20, 60, 80, 90, 75, 120, 110, 100, 70, 50

* To get sorted order of element from B.S.T. $O(n)$, just go for In-order.

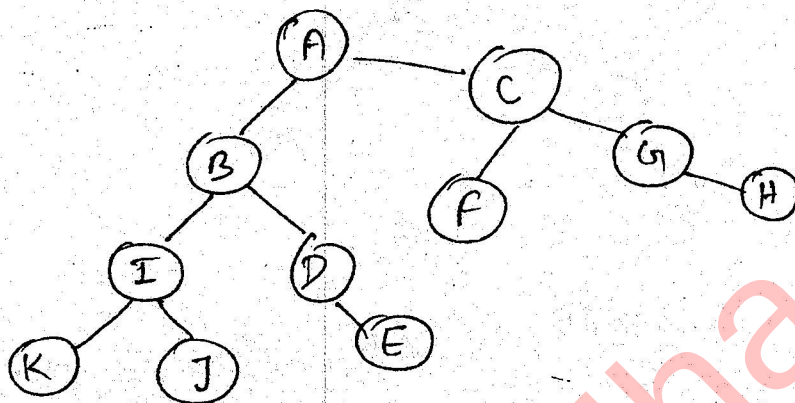
** Note :- In-Order traversal of B.S.T will always produce ascending order.

:- min Heap Tree will Take :- $O(n \log n)$ as $O(\log n)$ time for deletion

Que:- Consider the following Binary Tree data

Pre order: A B I K J D E C F G H

In-order: - ((K(I)(J)) B(D E)) A(F) C(G H))

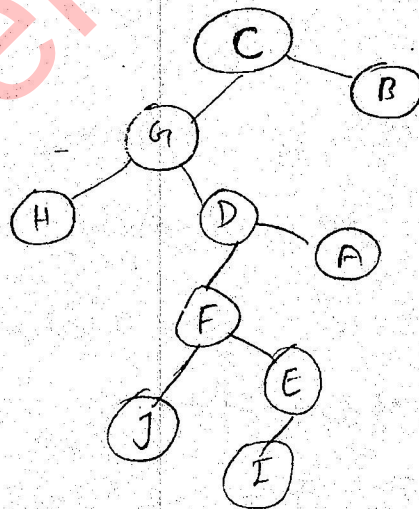


Post order: - K J I E D B F H G C A

Que:- Consider the following Binary Tree Data:

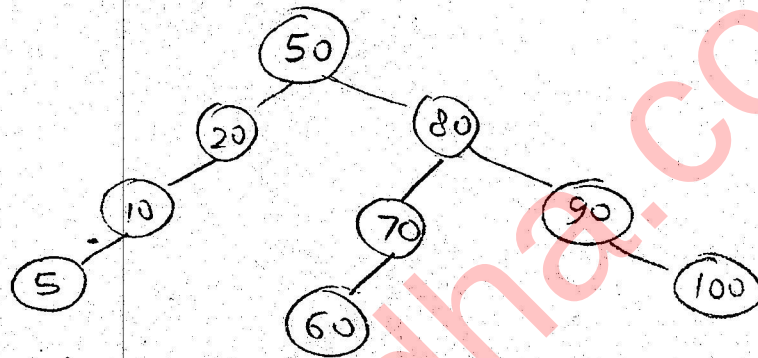
Post-order: H J I E F A D G B C

In-order: (H G J F (I E D A) C (B))



Note:- for the given In-order or In order, Post-order to construct Binary tree $O(n^2)$ time required. [n times Linear Search (worst case)]

Que:- Consider the following B.S.T. Data
Pre: 50, 20, 10, 5, 80, 70, 60, 90, 100



Post: 5, 10, 20, 60, 70, 100, 90, 80, 50

:- because of B.S.T., it is possible

B.S.T - In-order = Ascending order

In-order :- 5, 10, 20, 50, 60, 70, 80, 90, 100

$= O(n \log n)$ to sort (by merge sort)

**

:- As it is sorted, now we can apply Binary Search & its time = $n \times \log n \Rightarrow n \log n$

Total Time = $O(n \log n) + O(n \log n)$

= $O(n \log n)$ to Construct Binary Tree.

→ To Create B.S.T for the given Pre-order or Post-order will take $O(n \log n)$ time

[i) Sorting = $n \log n$

ii) Binary search (n times) = $n \log n$

$\Rightarrow O(2n \log n) \Rightarrow O(n \log n)$

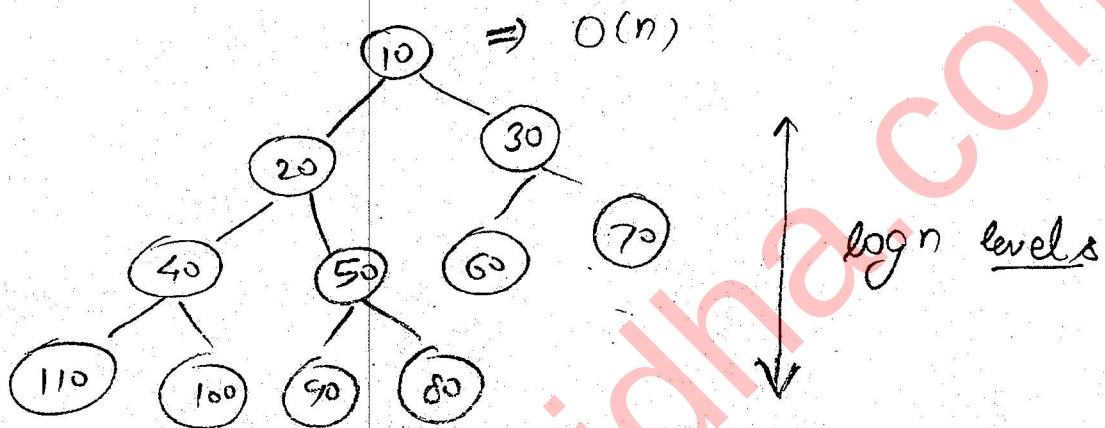
Ques :- Consider the following min-Heap Tree

Data :

In-order : $((110, 40, 100), (20, 90, (50, 80)), 10, 60, (30, 70))$

Post : ?

Ascending order :- $10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110$



~~Time~~ ~~To Post~~ :-

** Post-order :- $110, 100, 40, 90, 80, 50, 20, 60, 70, 30, 10$

'Time' :- By Linear search, to find out
min element $O(n)$ time
(as In-order not sorted.)

:- min Heap so $\log n$ levels Hence Time $\Rightarrow n * \log n$
 $= O(n \log n)$

** Note :- To construct a "unique Binary Tree"

In-order compulsory.

Pre	{	Binary Tree ✓	Post	{	Binary Tree ✓
In			unique B.T. ✓		

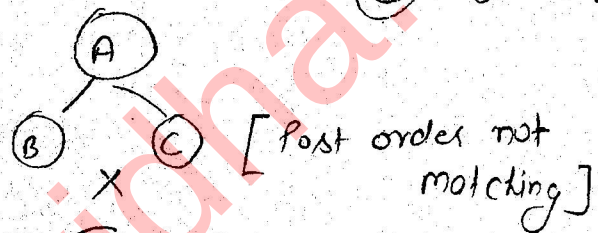
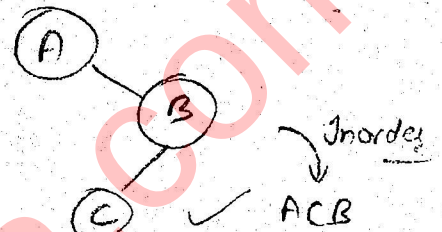
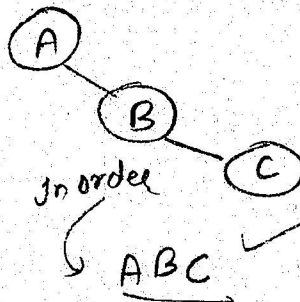
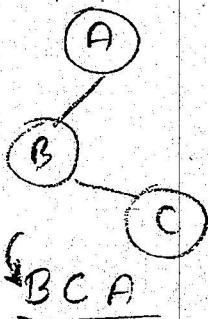
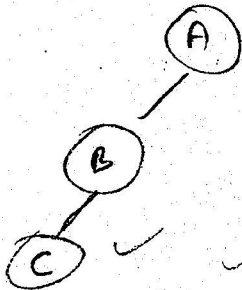
Que:- Consider the following Binary Tree data:-

Pre-order: A B C

Post-order: C B A

Inorder: ?

Right (In-order not given check manually (guess))
(many possibilities)

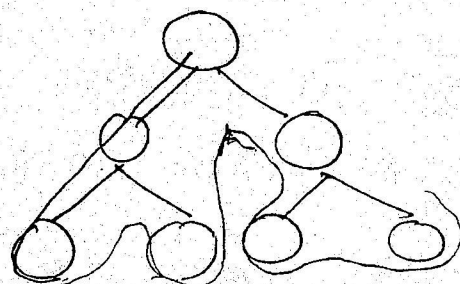
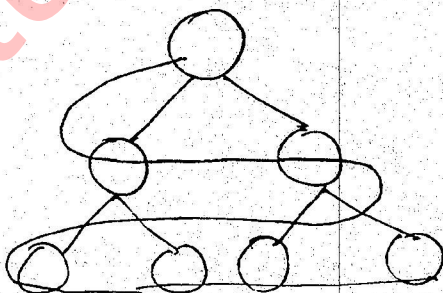


Pre } Binary Tree ✓
Post } unique Binary Tree (not possible)

"Graph Traversal" :-

1. Breadth first Traversal (BFT)
2. Depth first Traversal (DFT)

① "Breadth first Traversal" :-



BFT (V)

```
{ visited [v] = 1 ;  
  add (v,  $\phi$ ); // adding to Queue  
  while ( $\phi$  is not empty)
```

```
{  
  x = delete ( $\phi$ );  
  Pf(x)  
  for (all w adj to x)  
  {  
    if (w is not visited)  
    {  
      visited (w) = 1;  
      add (w,  $\phi$ );  
    }  
  }  
}
```

*:- graph Traversal is not unique, as in graph Traversal we can start traversing from any vertex. (So multiple DFT, BFT Possible)

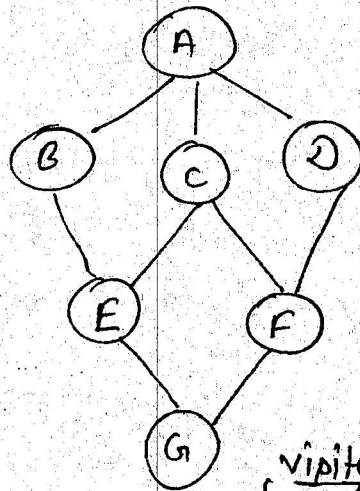
*:- Tree Traversal is always unique, as we have to start from Root always.

*:- Tree is always "Directed" and connected direction $\rightarrow$ Parent $\rightarrow$ children.

*:- Tree also called Directed acyclic graph (DAG).

*:- In Tree Traversal is unique so only one Pre-order, Post order, In-order.

Ex: ① :-



BFT :-

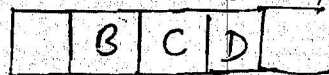
let's start with 'A'

Queue



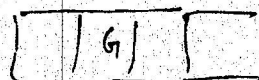
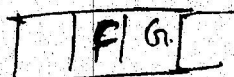
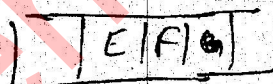
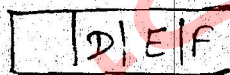
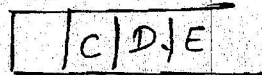
Delete 'A'

3 times for loop



Delete 'B'

2 times for loop (2 adj) (using adj list)



∴ total 2E time for loop
(sum of degree)

/* for loop is for adjacent, while loop is for vertices and they are dependent
 DO $\Rightarrow V + 2E$ */

:- To Implement B.F.T, we are using "queue"

Data Structure

:- Time Complexity = $V + 2E$ ^{To find adjacent total time}
 $\Rightarrow \underline{O(V+E)}$ using adj list.

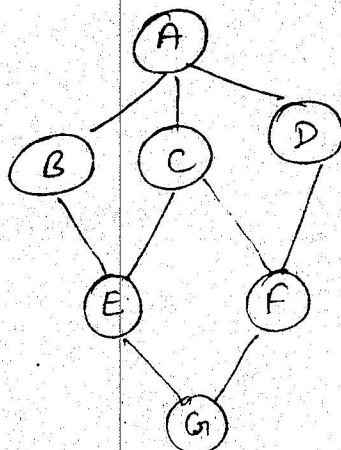
using adj matrix $\Rightarrow \underline{O(V^2)}$

(To find adjacent for one vertex = $O(V)$
for V vertices = $V * V = O(V^2)$)

* :- Space :-
Graph $\rightarrow$ adj list $\downarrow$ $V+E$
ip + Extra $\rightarrow$ Queue $\downarrow$ V
visited array $\downarrow$ V
 $\Rightarrow \underline{3V+E} \Rightarrow \underline{O(V+E)}$

* BFT is also known as 'level order Traversal'
printing level by level.

Que :- Consider the following graph :-



give four different BFT's :-

Soln:-

i) A B C D E F G
A B C D E F G

ii) G F E C D B A

iii) C A E F B D G

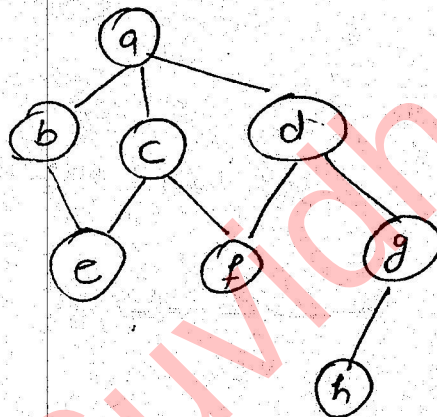
or
C E A F B G D

iv) D F A C G B E

/* A B C D E F G

These 3 can
come in any
order.

Que:- Consider the following Graph :-



Give 4 different BFT's starting from h.

i) h g d a f b c e

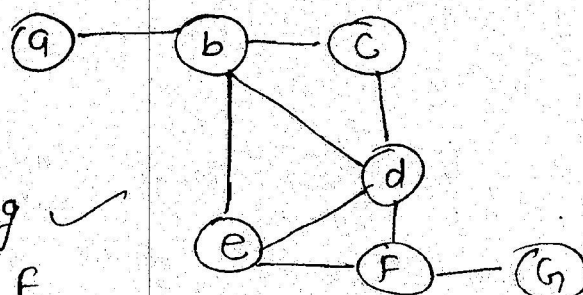
ii) h g d f a c b e

iii) h g d a f c b e

iv) h g d a f X (Not possible)

only "three"

Ex: ④ :-



1) a b c d e f g ✓

2) d b e c f X

3) b a c d e f X

4) a d e f b c g X

Applications of "BFT" :-

* 1.) we can verify, given graph is connected or disconnected.

* 2.) we can find out number of connected components.

:- Connected Component :- A maximal subgraph which is connected.

Time (To find # of connected components) = $O(V+E)$
(because, only once we visit edges and vertices as all are disconnected)

* 3.) using BFT, we can verify, given graph contain cycle or not.

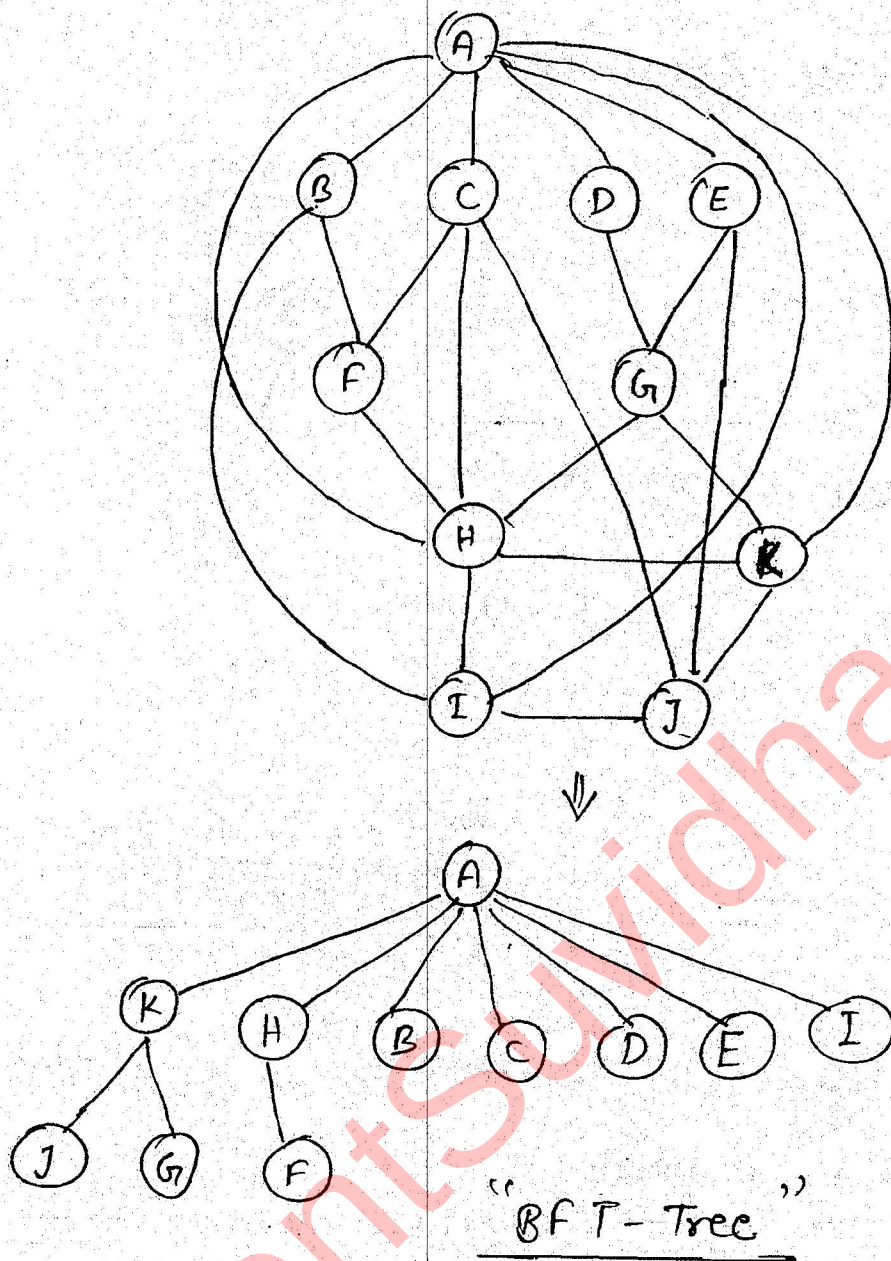
:- In kruskal Algo. cycle checking is done by BFT in background which will take

$O(V+E)$ time

$$\text{So kruskal Time} = E \log E + V + E \\ \Rightarrow O(E \log E)$$

* 4.) we can verify, given graph is Bipartite graph or not.

* 5.) using BFT, we can find out, single source shortest path in the given unweighted graph.



∴ * BFT is also used for finding "Spanning Tree" of graph.

* If graph has all positive weights and equal weight then "BFT" algo is used for single source shortest path.

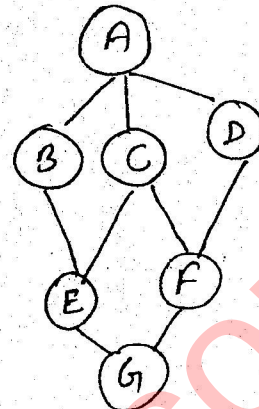
Date
23/10/2017

"Depth first Search" :-

DFT (V)

1. visited [v] = 1
2. Pf (v);
3. for (all w adj to v)

If (w is not visited)
 DFT (w);



visited						
1	1	1	1	1	1	1
0	0	0	0	0	0	0
A	B	C	D	E	F	G

lets start: with A



Print it out:- A B E G F D C

DFT (A)

A adj to stack

- 1
- 2

3. w = B C D

1. B DFT (B)

- 1
- 2
3. A E
1. A (visited)
2. E

DFT (E)

- 1
- 2
3. w = B G C
2. G

DFT (G)

- 1
- 2
3. w = A F

DFT (F)

- 1
- 2
3. w = A D C

DFT (C)

- 1
 - 2
 3. w = A B E
- pop

$\Leftarrow$ DFT (D)
 1.
 2.
 3. $w = AF$
 1. ~~X~~
 2. ~~X~~ Pop

} so 7 function calls
 but only 6 stack
 units needed.
 Pop also considered as
 Backtrack here.
 :- Pop also 7 times
 (Backtrack) (in b/w ones)

Time :- for loop :- $O(2E)$
 $\Rightarrow O(E)$

:- Recursion :- 7 function call or
 ($\sqrt{}$ function calls) $\Rightarrow O(\sqrt{V})$

Time complexity = $O(V+E)$

* i) we are using "Stack" data structure.

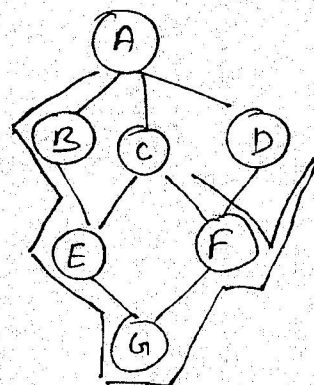
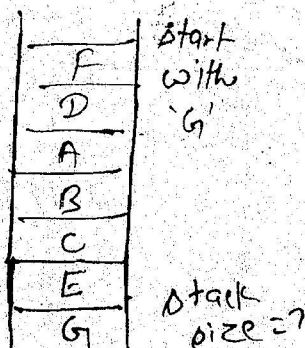
* ii) Time complexity = $O(V+E)$ (using Adj list)
 $= O(V^2)$ (using Adj matrix)

* (ii) Space complexity = $\frac{V}{p} + \frac{\text{Extra}}{\text{Adj list (V+E)}}$
 $\frac{V}{p} + \frac{\text{Visited (Array)}}{V} + \frac{\text{Stack}}{V}$
 $\Rightarrow 3V + E \Rightarrow O(V+E)$

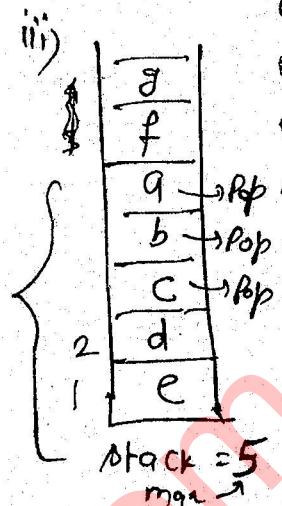
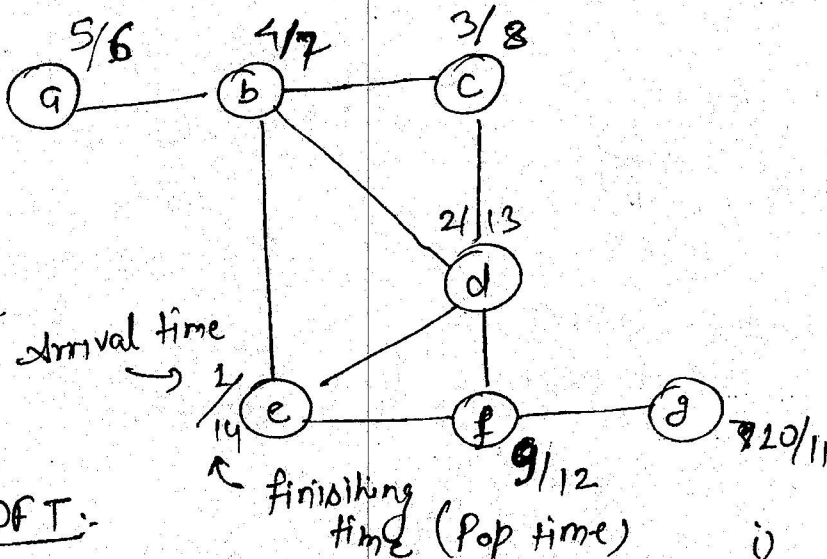
Que :- Consider the following graph :

[give 4 different BFT'S]

- i) A B E G F D C —
- ii) G E B A D F C —
- iii) A D F G E C B —
- iv) C E G F D A B —
- v) F C A B D G E X

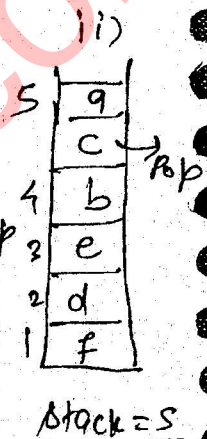
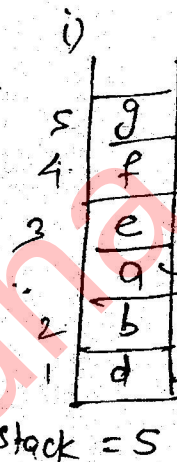


Ex(3):-

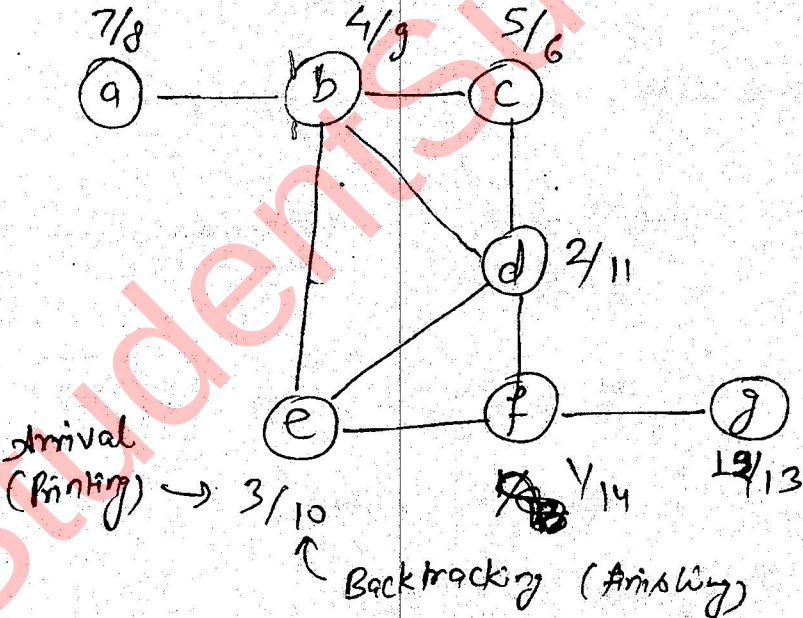


T/f - DFT:-

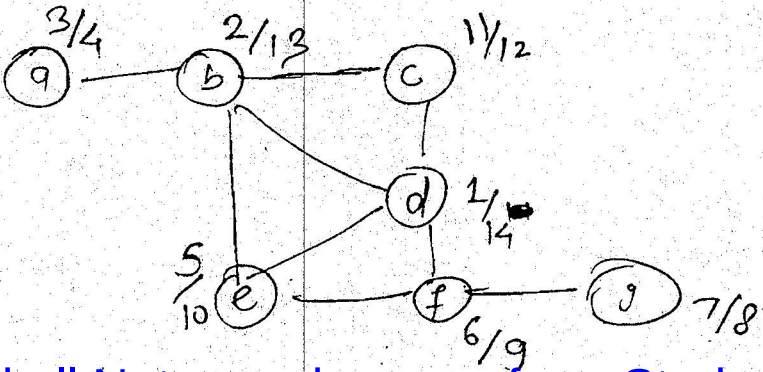
- i) d b a e f g c ✓ → Stack
- ii) f d e b c a g ✓
- iii) c d b a f e g X
- iv) e d c b a f g ✓
- Below (iv) are fractions: $\frac{1}{14}, \frac{2}{13}, \frac{3}{8}, \frac{4}{7}, \frac{5}{6}, \frac{9}{12}, \frac{10}{11}$



ii)



i)



Applications of DFT :-

* i) we can verify, given graph is connected or not
(Time : $O(V+E)$)

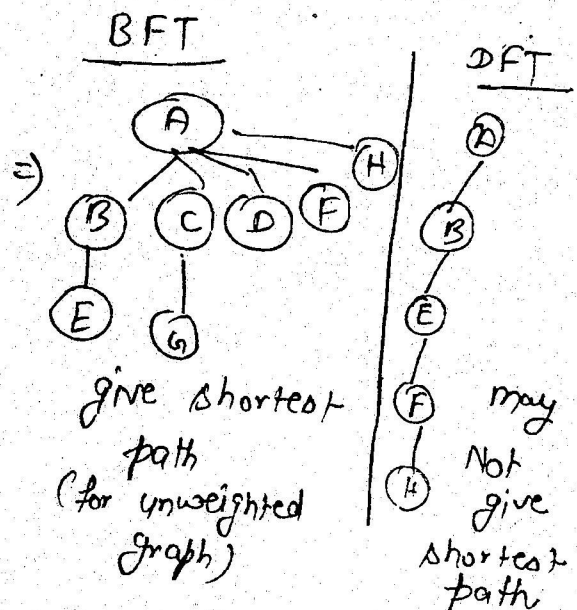
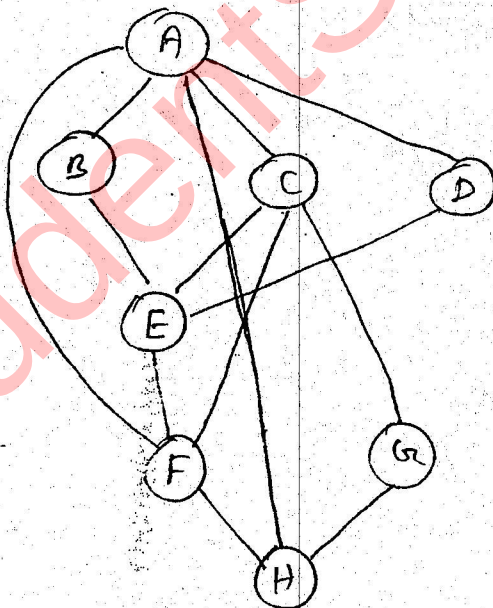
* ii) we can findout # of connected components
Time : ($O(V+E)$) [4 times DFT but Complete scan is only once]

* iii) we can verify, the given graph contain cycle or not.

:- To findout cycle in linklist or finite automata apply BFT / DFT.

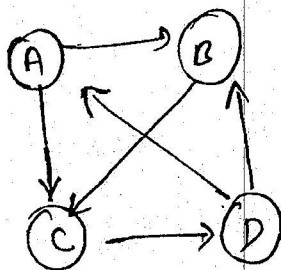
* iv) using DFT, we can findout single source shortest path in the given unweighted graph.

Ex



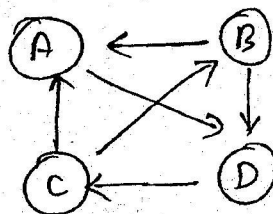
* v) using DFT we can verify given directed graph is strongly connected or not.

Ex:-



Apply DFT:- A C D B

:- Reverse direction
then Apply DFT



DFT:- A D C B

:- (all vertices come for both time, so there is the path between every two vertices.)

:- So the graph is Strongly connected.

Two times DFT :- ~~$O(2(V+E))$~~ $\Rightarrow O(V+E)$

*vi)

:- Articulation Point :-

→ A vertex is said to be articulation point iff By deleting that vertex together with its edges, Resultant graph is disconnected.

→ using DFT, we can verify if a vertex is articulation point or not.

→ for checking, for one vertex for articulation point, we have to apply DFT twice $\Rightarrow O(V+E)$