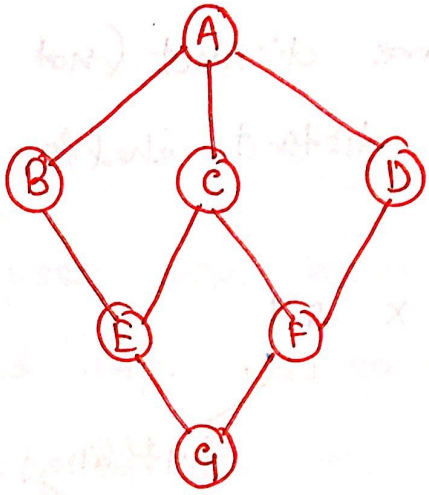
⇒ Graph Traversal :- (BFT & DFT)



Visited vertices array :-

not visited

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

- once an ~~~~ vertex is visited, corresponding value of A in array is set to 1.

Adjacency matrix

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

$O(V \times V) = O(V^2)$

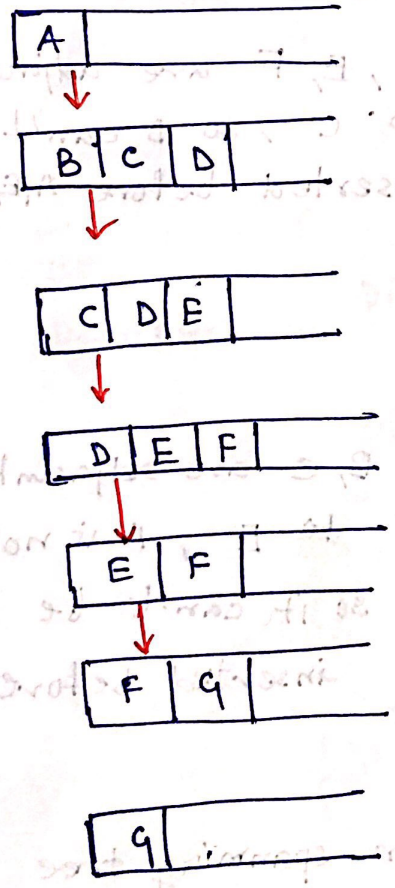
Adjacency list

A	B, C, D
B	A, E
C	A, E, F
D	A, F
E	B, C, G
F	C, D, G
G	E, F

Implement-ation complexity $O(V+E)$

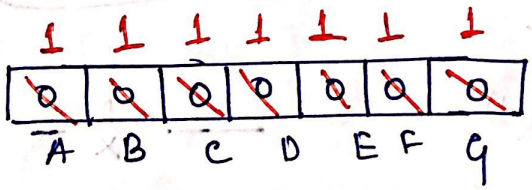
(1) Breadth First ~~Search~~ Traversal (BFT) :-

- Take any vertex (let A), and insert it in array (Queue)
- Make corresponding array element in visited array = 1.
- Once it visited, delete it and add its adjacent element (not visited) inside the queue.
- Repeat all these steps until all vertices are visited. (all array elements become 1).



Queue

when queue empty, BFS stopped



visited vertices array

NOTE:- It is implimented using Queue.

It is also known as level-order traversing

Time complexity	BFS	
	Adjacency matrix	Adjacency List
	$O(V^2)$	$O(V+E)$

- Graph → more than one BFS or DFS possible (due to cycle)
- Tree → Traversing is unique (no cycle)

Q. Which of the following are possible BFT :

(a) A B C D E F G ✓

(b) A C B D E F G ✓

(c) G E F B C D A ✓

(d) C A E D F G ✗

A, E, F are adjacent to C, so D can't be inserted before A, E, F.

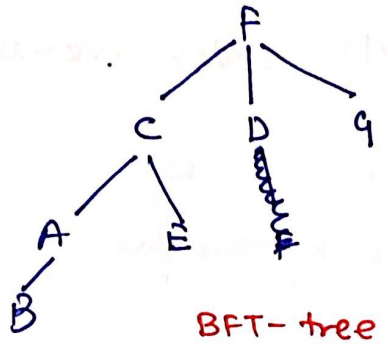
(e) F C D G A E B ✓

(f) G E F A B C D ✗

B, C are adjacent to E, A is not so it can't be inserted before B, C.

NOTE :- BFS will give BFT – tree or spanning tree for a graph.

∴ F, C, D, G, A, E, B ∴



BFT- tree

* BFT- tree is just spanning tree (not minimum spanning tree)

NOTE:-

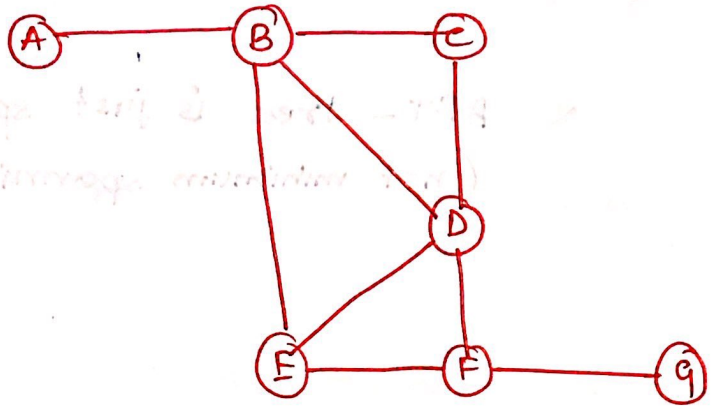
Graph ⟶ Minimum spanning tree (Kruskal's or Prims algo)

O((V+E) log V)

Graph ⟶ spanning tree: (BFS) O(V+E)

	Time complexity	Space complexity
BFS algorithm	O(V+E) adjacency list O(V²) adjacency matrix	Space → I/p (V+E), Extra → Queue (V), Visited array 3V + E O(V+E)

Q.



Which of the following BFS order are valid:

(a) ~~D B C E F G A~~

(b) A B C D E F G

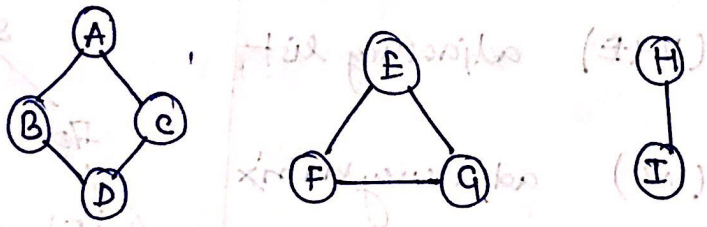
(c) ~~G F E D C B A~~

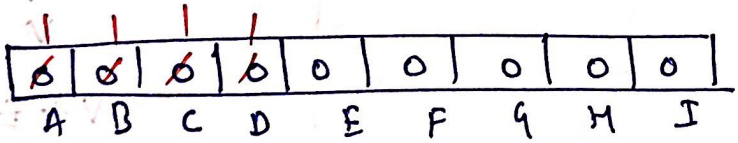
(d) ~~B A C D F E G~~

Application of BFT :-

(i) To check whether graph is connected or disconnected.

eg:-





all are not visited so, graph is disconnected.

(ii) Can Find out no. of connected components (maximum subgraph connected) of a graph.

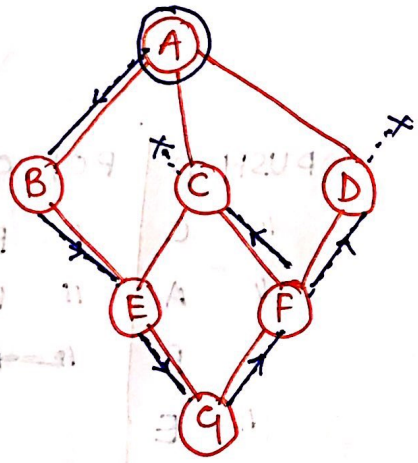
$O(V+E)$ (K = no. of times BFT applied for visit all vertices.)

(iii) Using BFT, we can verify given graph contain cycle or not.

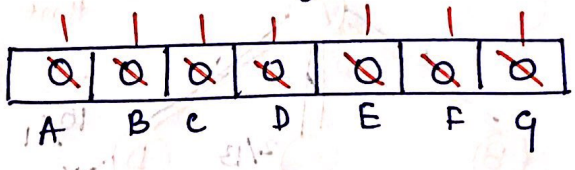
(iv) we can findout single source shortest path in given unweighted graph.

(v) we can verify given graph is Bipartite or not.

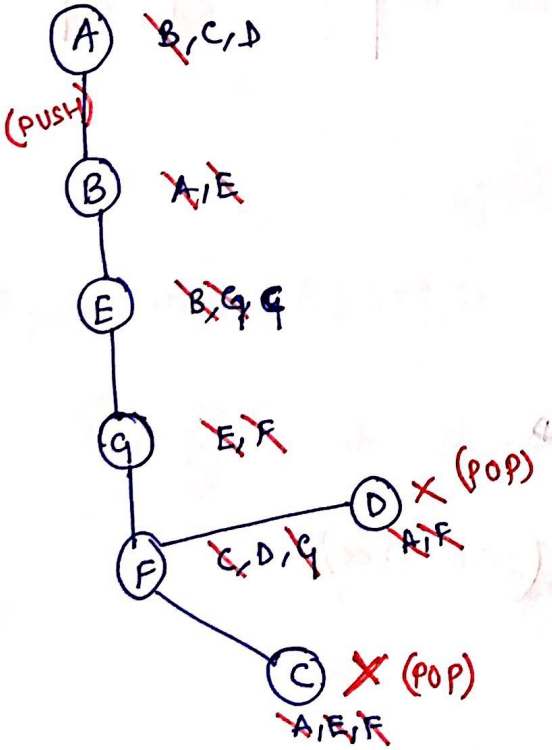
(2) Deapth First Traversal :-



- Visited array required -

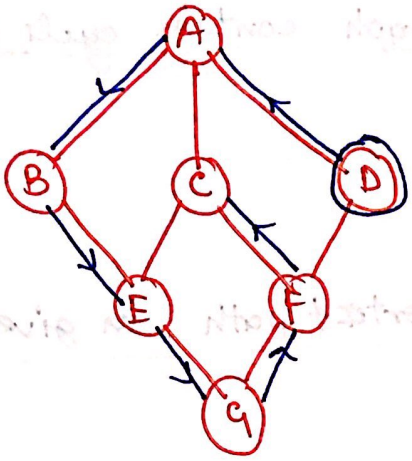


- stack is used for implementation.



{A B E G F C D}

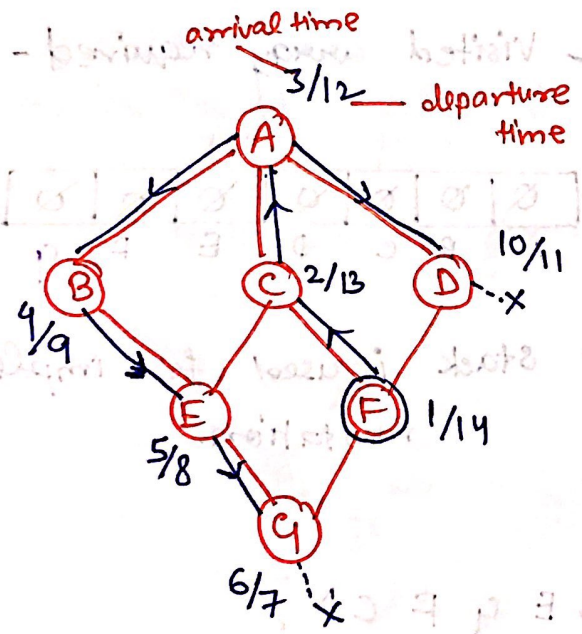
PUSH A	POP F
" B	" G
" E	" E
" G	" B
" F	" A
" C	
POP C	
PUSH D	
POP D	



PUSH	POP
PUSH D	POP C
// A	// F
// B	// G
// E	// E
// G	// B
// F	// A
// C	// D

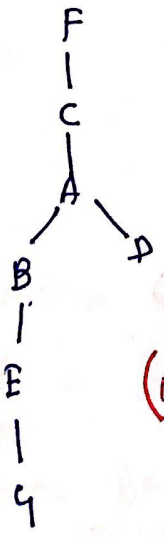
{D A B E G F C} — continuous PUSH and then POP (it will use max STACK size)

- If we have in b/w PUSH & POP op^n then we can not use max stack size.



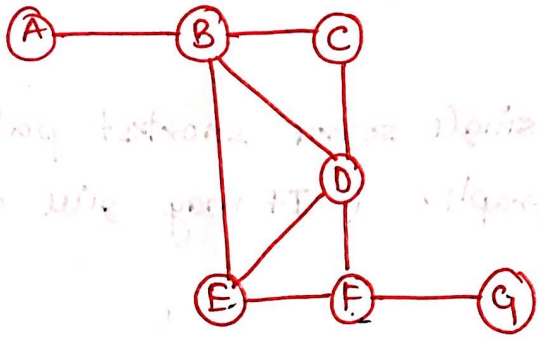
PUSH	POP	PUSH	POP
// C	// E	D	POP D
// A	// B		POP A
// B			POP C
// E			POP F
// G			

{F C A B E G D}



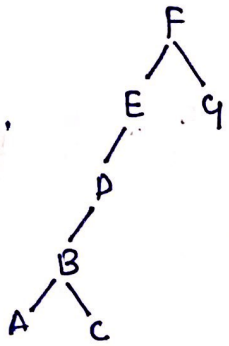
(DFT-tree)

Q.



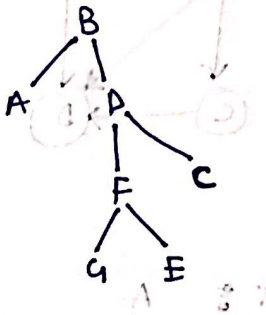
which of the following DFT are valid:

(a) F E D B A C G



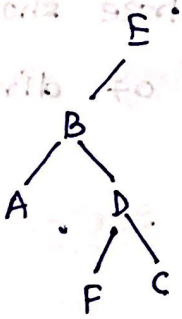
(stack size = 5)

(b) B A D F G E C



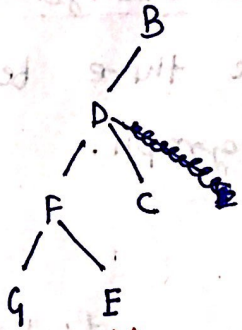
(S = 4)

(c) E B A D F C G



· G not reachable

(d) B D F G C E A



Applications of DFT:-

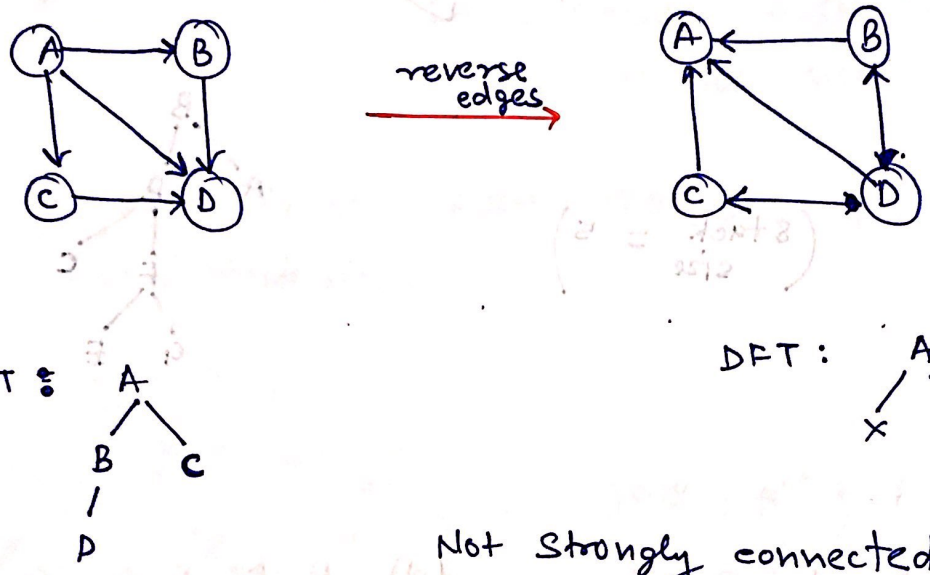
(i) verify graph is connected or not

(ii) Findout no. of connected componerited in a given graph.

(iii) Verify given graph contain cycle or not.

(iv) we can't find out single source shortest path in given unweighted graph. (It may give or may not)

(v) Verify graph is strongly connected or not.

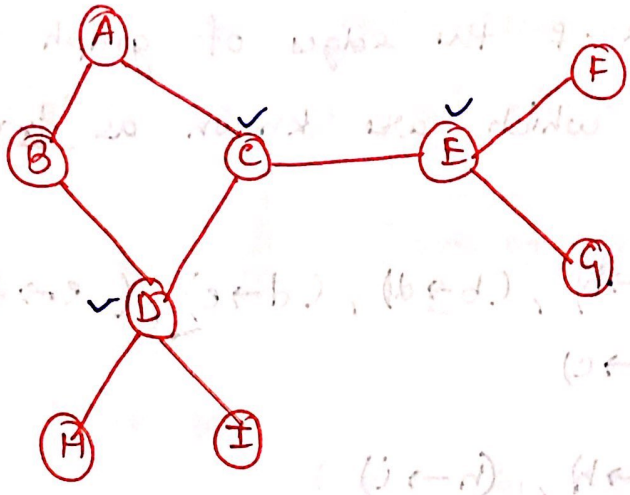


Not Strongly connected.

* For strongly connected graph, DFT - tree should be there before and after reversing of directed graph.

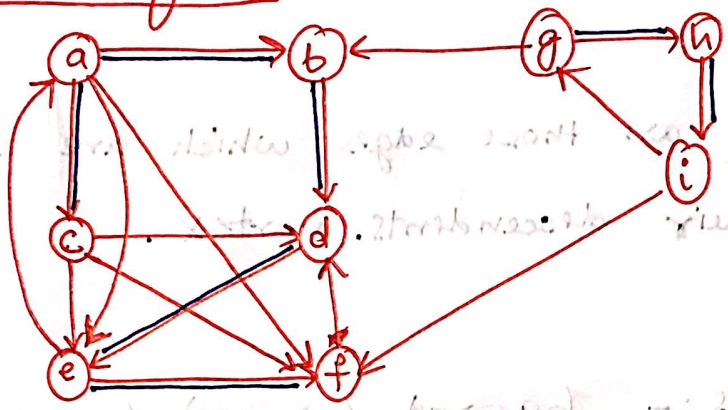
(vi) We can find out no. of articulation points present in given graph.

Articulation point : By deleting any vertex with its edges and graph becomes dis-connected.



(D, C, E articulation points)

Types of Edges :-

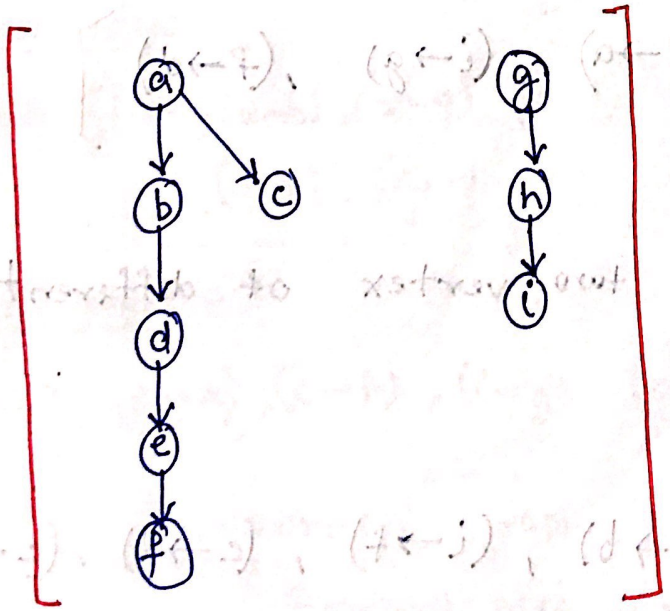


Make DFT-tree for this graph.

(DFT — directed) :-

Since it is a graph so more than one DFT-tree are possible:

DFT-tree



Since K=2 so 2 components.

- As we can see, some of the edges of graph are only used in Tree which are known as "tree edges".

[Tree edges :- (a→b), (b→d), (d→e), (e→f), (a→c), (g→h), (h→i)]

- "Forward edges" are those edges which are connecting ancestor to their descendents. vertex.

[Forward edges :- (a→e), (a→f)]

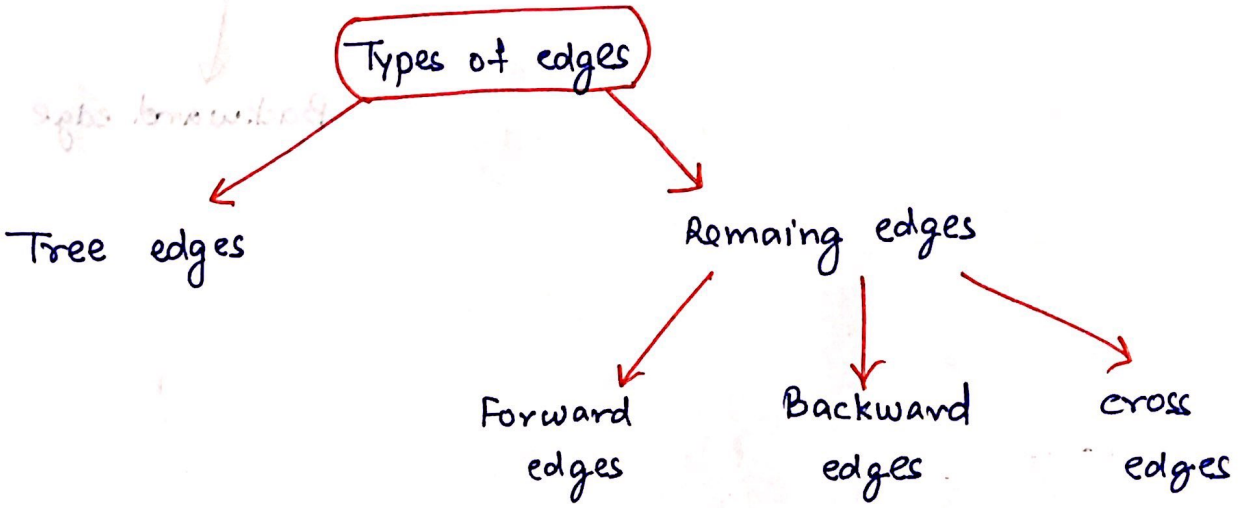
- "Back edge" are the edges which connects descendent vertex to their ancestor vertex.

[Back edges :- (e→a), (i→g), (f→d)]

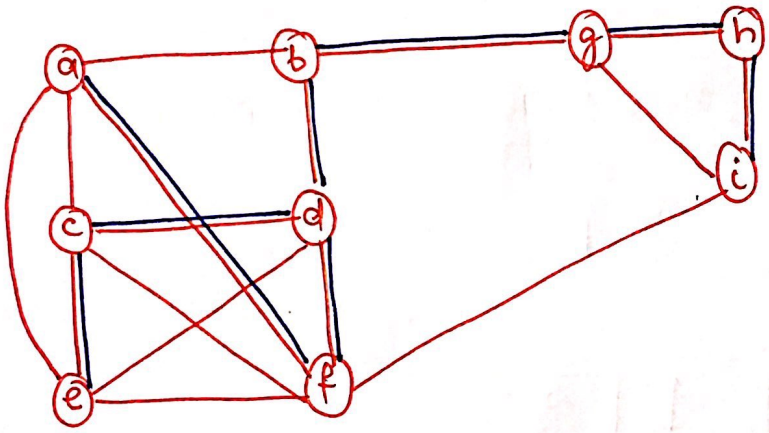
- "Cross edge" connects two vertex of different family

[Cross edge :- (g→b), (i→f), (c→d), (c→f)]

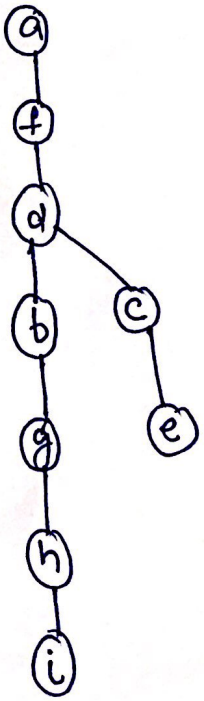
NOTE:- Ancestor and descendant vertex relationships are find out on the basis of tree.



(DFT- undirected):-



DFT- tree



Tree edges:-

(a-f), (f-d), (d-b), (b-g), (g-h), (h-i)
(d-e), (c,e)

Back edges:-

(c-a), (e-a), (c-f), (i-g), (b-a), (e-d), (i-f)

NOTE:- In undirected graph **Back-edge & Forward edge are same** and **cross edge are not possible.**

Types of edges

- Tree edges
- Remaing edges
  - Backward edge