Graph Traversal Algorithms:

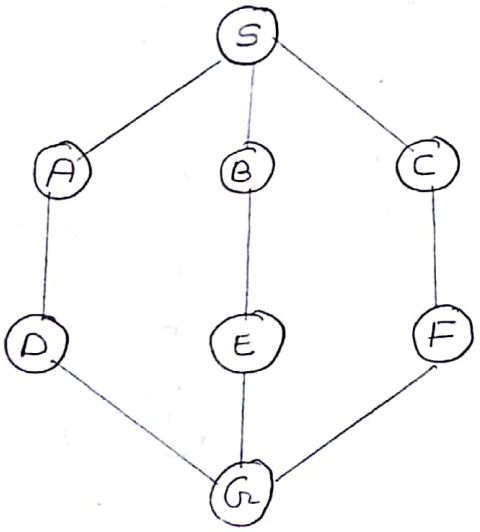
Depth First Search:

is an algorithm for traversing or searching tree or graph data structures.

One starts at root (selecting some arbitrary node as the root in case of a graph) and explores as far as possible along each branch before backtracking.

Examples: Perform DFS on:

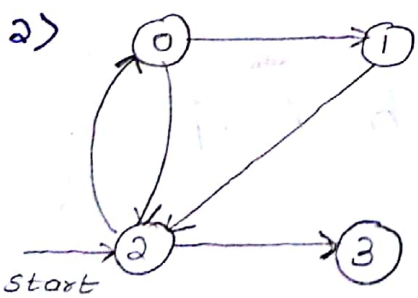
1>



Traversal:

S A D G E B F C

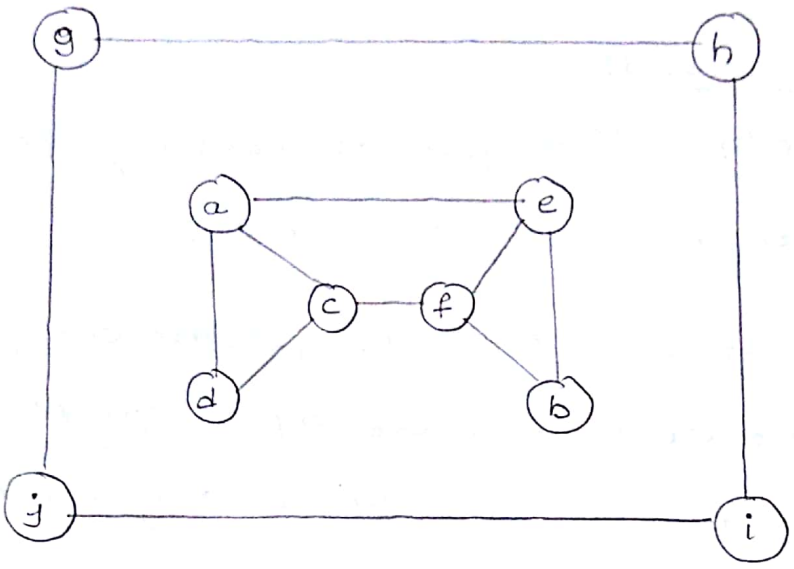
2>



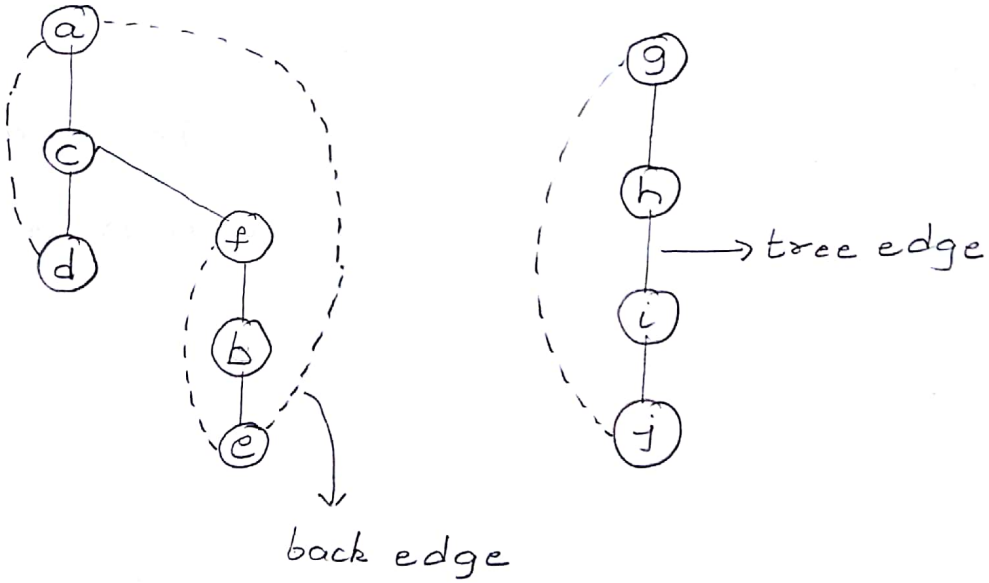
Traversal:

2 0 1 3

3)



DFS forest:



Traversal result:

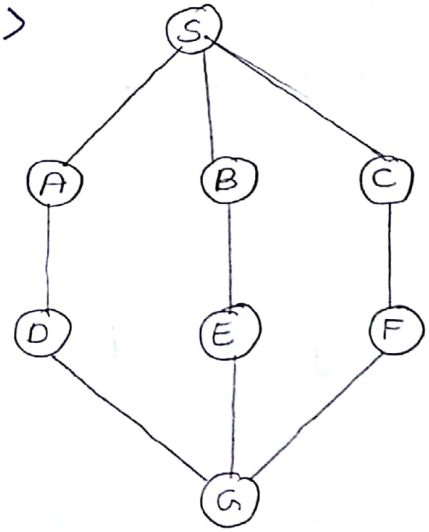
a c d f b e g h i j

Breadth First Search:

is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key') and explores the neighbor nodes first, before moving to the next level neighbors.

Examples: Perform BFS on:

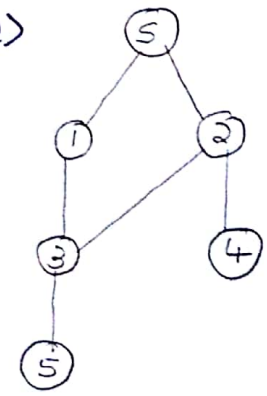
1>



Traversal:

S A B C D E F G

2>

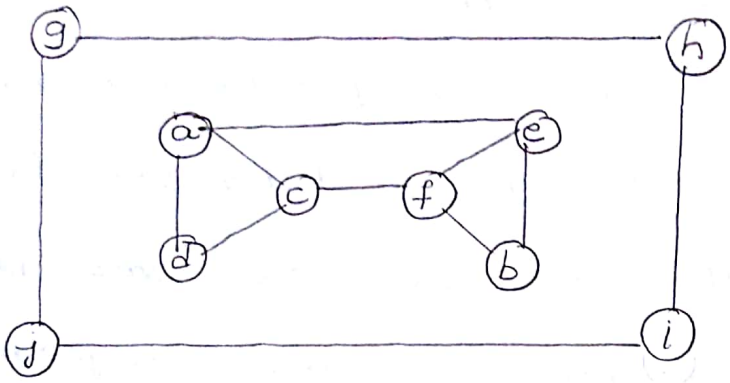


Traversal:

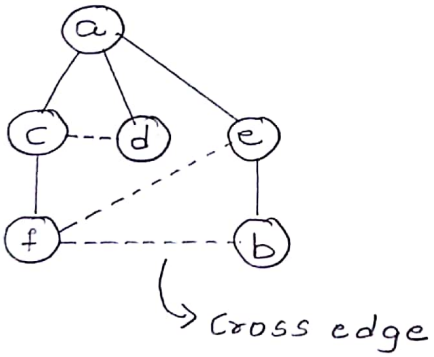
S 1 2 3 4 5

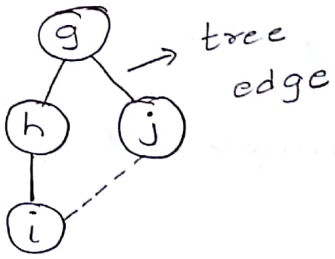
-PH

3)



BFS Forest:





Traversal result:

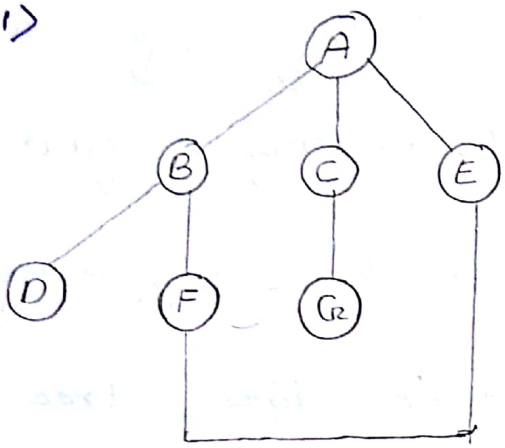
a c d e f b g h j i

DFS and BFS Comparison:

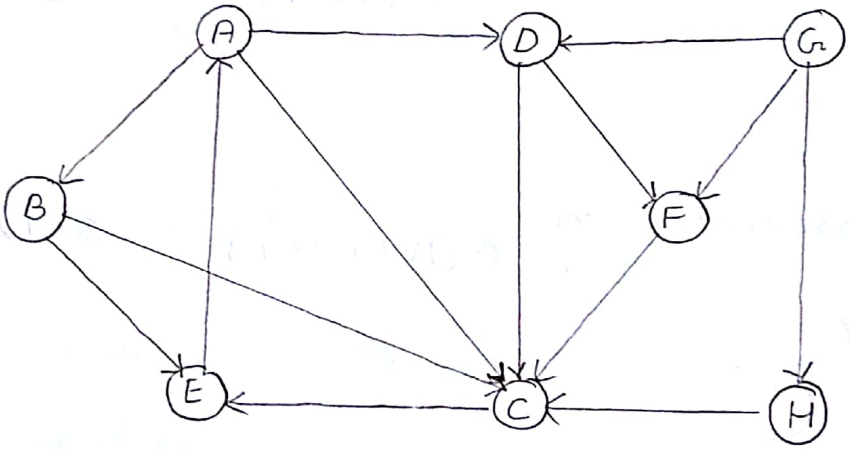
Feature	DFS	BFS
1. For	Brave ones	cautious ones
2. Data structure	Stack	Queue
3. Edge types	tree edges back edges	tree edges cross edges
4. Matrix representation efficiency	$\Theta(\|V\|^2)$	$\Theta(\|V\|^2)$
5. List representation efficiency	$\Theta(\|V\|+\|E\|)$	$\Theta(\|V\|+\|E\|)$
6. Vertex Orderings number	2 orderings	1 ordering
7. Applications	-Detecting cycles -Path finding -Bipartite graph -maze with one solution -etc	- Crawlers - Social n/w websites - GPS systems -Cycle detection -etc

Exercise : Perform DFS & BFS on:

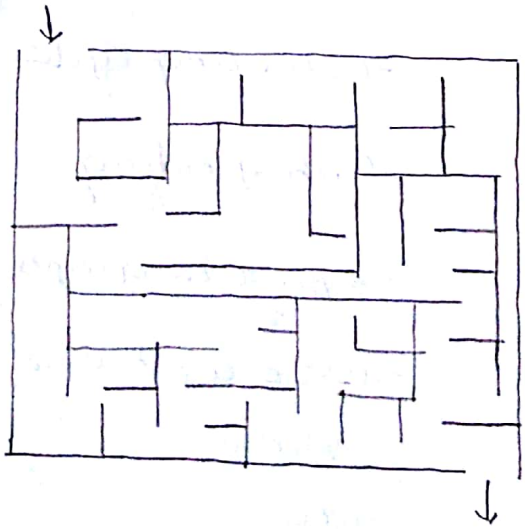
1)



2)



3) Is the maze for brave ones or cautious ones?



Generic Graph Search:

Goals:

a) Find everything reachable from given start vertex

b) Dont explore anything twice

Generic Algorithm: Given graph G, vertex s

- initially s-explored, all other vertices unexplored
- while possible:
  - choose an edge (u, v) with u explored & v unexplored
  - mark v explored

{ if none, halt

Discovering all the Nodes:

Input Graph (s)

DiscoveredNodes ← {s}

while there is an edge e leaving DiscoveredNodes that has not been explored:

add vertex at other end of e to the DiscoveredNodes

return DiscoveredNodes

-PH

Book-keeping:

- How to keep track of which edges / vertices we have dealt with?
- What order do we explore new edges in?
- What if edges have weights?
- Can we sort the edges?
- Is the graph directed or undirected?
- How do we represent the graph?
- Does the graph have more than one component?
- Is the graph sparse or dense?
- Do we need to keep track of discovered edges or discovered vertices?
- How do we make sure that every node or is visited only once?
- What other supporting Data Structures would be needed for traversal?

Preorder & Postorder Numbers your DFS:

DFS marks all vertices as visited. We can certainly keep track of other data that can be useful. Design Augment functions to store additional information.

```
Explore(v)
visited(v) ← true
previsit(v)
for (v, w) ∈ E:
    if not visited(w):
        explore(w)
postvisit(v)
```

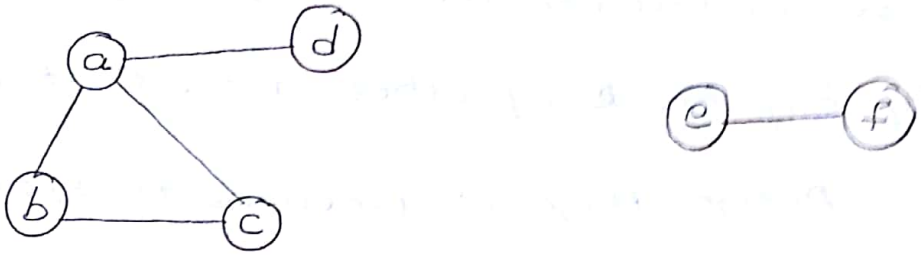
What can be previsit(v) & postvisit(v)?

One possible option:

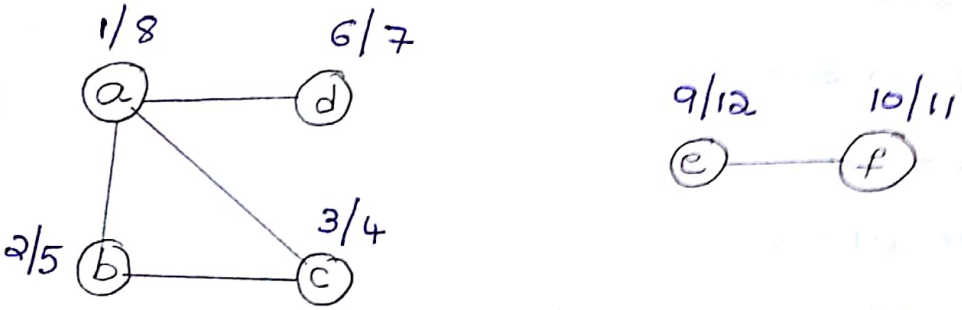
- keep track of order of visits
- Record previsit & postvisit times for each v.
- Its like having sort of clock ticks

we can certainly design the functions any other way as the application demands.

Example:



Ordering:




```
tick ← 1
previsit (v)
  pre(v) ← tick
  tick ← tick + 1
postvisit (v)
  post(v) ← tick
  tick ← tick + 1
```

Previsit and Postvisit numbers tell us about the execution of DFS.

Lemma: For any vertices u, v the intervals [pre(u), post(u)] and [pre(v), post(v)] are either nested or disjoint.

Nested & Disjoint:

Nested :

Disjoint :

The interleaved case is not possible:

Which of the following tables is not a valid set of pre- & post- orders?

Table 01 :

V	Pre	Post
A	1	8
B	9	10
C	3	4
D	2	7
E	5	6

Table 02 :

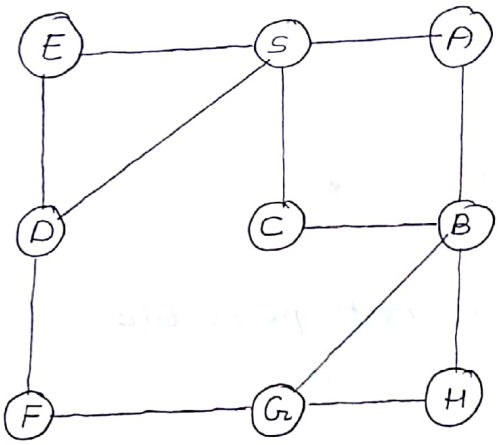
V	Pre	Post
A	1	9
B	8	10
C	2	7
D	3	6
E	4	5

Answer : Table 02.

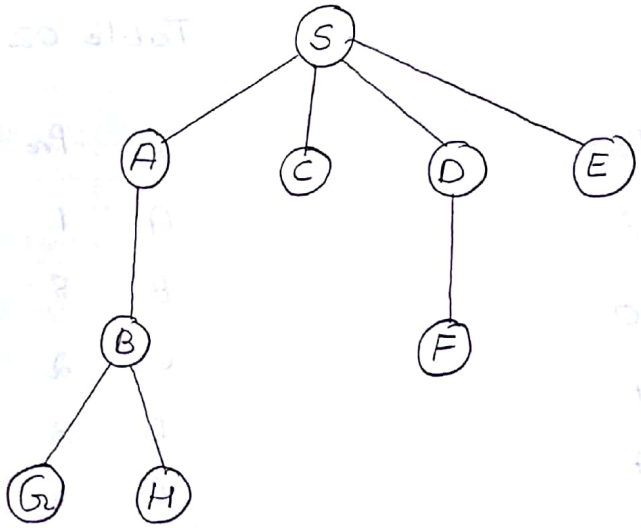
Can you justify why?

- PH

Question: Draw a shortest path tree for the given graph



We can apply BFS traversal to obtain the shortest path tree.



Note how BFS creates a tree with shortest path from given source vertex to every other vertices.