SCS1201 Advanced Data Structures

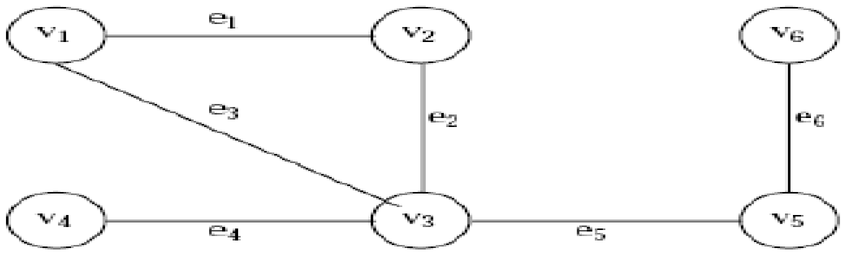
UNIT: 3 GRAPHS

Graph: A graph G is a defined as a set of objects called nodes and edges.

G= (V, E) , Where V is a finite and non empty set at vertices.

E is a set of pairs of vertices called edges. Each edge '*e*' in E is identified with a unique pair (*a*, *b*) of nodes in V, denoted by *e* = [*a*, *b*].

Example:



Consider the above graph 'G'. Then the vertex V and edge E can be represented as:

Vertex V = {v1, v2, v3, v4, v5, v6}

E = {e1, e2, e3, e4, e5, e6}

E = {(v1, v2) (v2, v3) (v1, v3) (v3, v4), (v3, v5) (v5, v6)}.

There are six edges and vertex in the graph

Node: A node is a data element of the graph.

Edge: An edge is a path between two nodes.

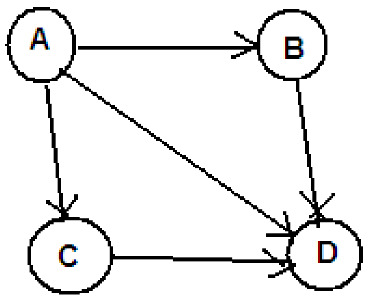
TYPES OF GRAPH

There are two types of graph. They are

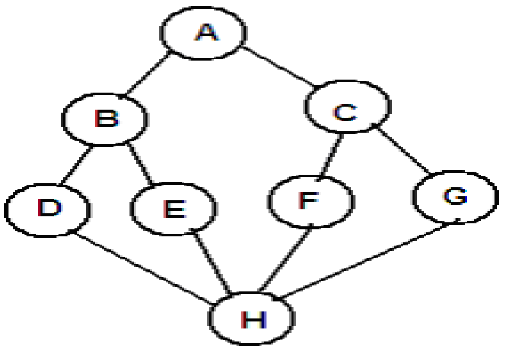
1. Undirected graph
2. Directed graph

Undirected graph: An undirected graph is a graph in which the edges are not directionally oriented towards a node.

Directed graph: A Directed graph or a Digraph is a graph in which the edges are directionally oriented towards any node.



Directed Graph



Undirected Graph

Weighted and Unweighted

Graphs can be classified by whether or not their edges have **weights**.

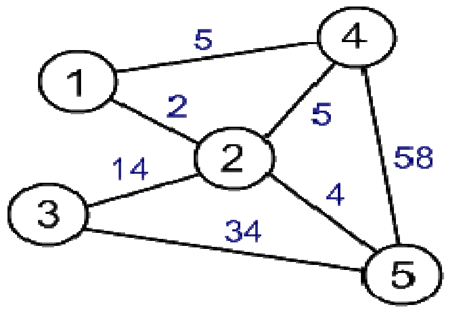
Weighted graph: Edges have a weight

- Weight typically shows cost of traversing.
- Example: weights are distances between cities

Unweighted graph: Edges have no weight

- Edges simply show connections

Example:



Distance between 1 and 4 is 5.

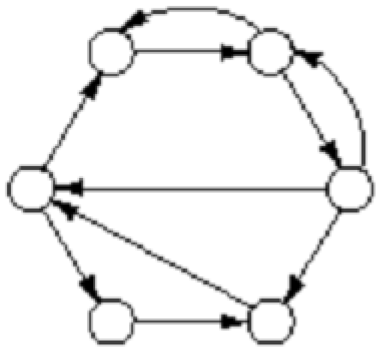
Distance between 3 and 5 is 34

BASIC TERMINOLOGIES

Arc: The directed edge in a directed graph is called an arc.

Strongly connected graph: A directed graph is called strongly connected if there is a directed path from any vertex to any other vertex

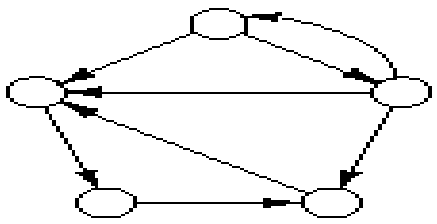
Example:



In the above graph we have path from any vertex to any other vertex

Weakly connected graph: A Directed graph is called a weakly connected graph if for any two nodes I and J, there is a directed path from I to J or from J to I.

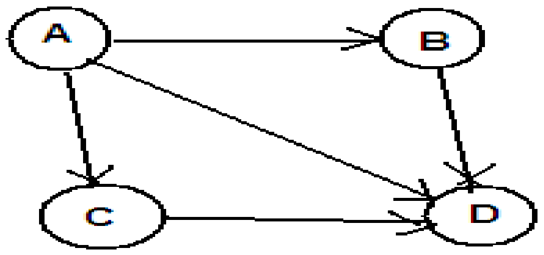
Example:



Out degree: The number of arcs exiting from the node is called out degree of that node.

In degree: The number of arcs entering the node is called in degree of that node.

Example:



Directed Graph

Nodes	Indegree	OutDegree
A	0	3
B	1	1
C	1	1
D	3	0

Source node: A node where the indegree is 0 but has a positive value for outdegree is called a source node. That is there are only outgoing arcs to the node and no incoming arcs to the node.

Example:

Node ‘A’ is the source node.

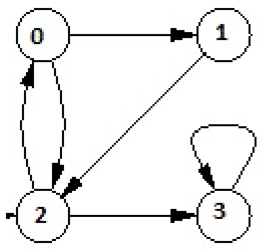
Sink node: A node where the outdegree is 0 and has a positive value for indegree is called the sink node. That is there is only incoming arcs to the node and no outgoing arcs the node.

Example:

Node ‘D’ is the Sink node.

Cycle: A cycle in a directed graph is a directed path that originates and terminates at the same node ie some number of vertices connected in a closed chain.

Example:



This graph contains three cycles 0->2->0, 0->1->2->0 and 3->3,

Degree of a node: In an undirected graph, the degree of a node is the number of edges connected directly to the node.

Length of the path: The length of the path between node I and K is the number of edges between them in a path from I to K.

Degree: The degree of the node B in the undirected graph shown above is 3.

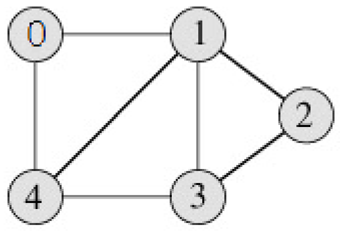
REPRESENTATION OF GRAPHS

There are two possible ways by which the graph can be represented.

1. Matrix representation (Array Representation)
2. Linked representation

The graphs can be represented using **Adjacency matrix** or otherwise called the **incidence matrix**. Since the matrix is so sparse it is also called as sparse matrix.

The adjacency matrix is a N X N matrix where N is the number of nodes in the graph. Each entry (I, J) in the matrix has either 1 or 0. An entry 1 indicates that there is a direct connection from I to J. An entry 0 indicates that there is no direct connection from I to J.



If an adjacency matrix is written for the above directed graph as shown:

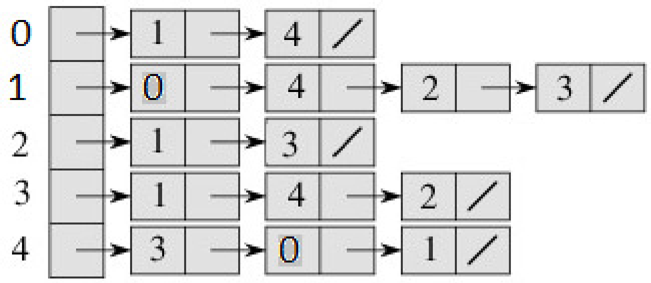
$$\begin{array}{c|ccccc|}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 \\
2 & 0 & 1 & 0 & 1 & 0 \\
3 & 0 & 1 & 1 & 0 & 1 \\
4 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}$$

Adjacency Matrix Representation

Since the matrix is so sparse in nature the second method of representation will be preferred if the number of edges is very less compared to the number of vertices.

Adjacency List:

An array of linked lists is used. Size of the array is equal to number of vertices. Let the array be array[]. An entry array[i] represents the linked list of vertices adjacent to the ***i***th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be stored in nodes of linked lists. Following is adjacency list representation of the above graph.



Adjacency List Representation of the above Graph

GRAPH TRAVERSALS

There are two methods for traversing through the nodes of the graph. They are:

1. *Breadth First Search Traversal (BFS)*
2. *Depth First Search Traversal (DFS)*

Breadth First Search Traversal (BFS)

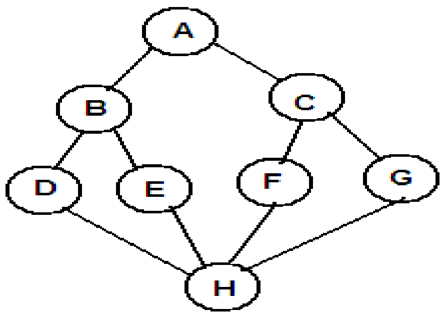
As the name implies, this method traverses the nodes of the graph by searching through the nodes breadth-wise. Initially let the first node of the graph be visited. This node is now considered as node u. Now find out all the nodes which are adjacent to this node. Let all the adjacent nodes be called as w. Add the node u to a queue. Now every time an adjacent node w is visited, it is added to the queue. One by one all the adjacent nodes w are visited and added to the queue. When all the unvisited adjacent nodes are visited, then the node u is deleted from the queue and hence the next element in the queue now becomes the new node u. The process is repeated on this new node u. This is continued till all the nodes are visited.

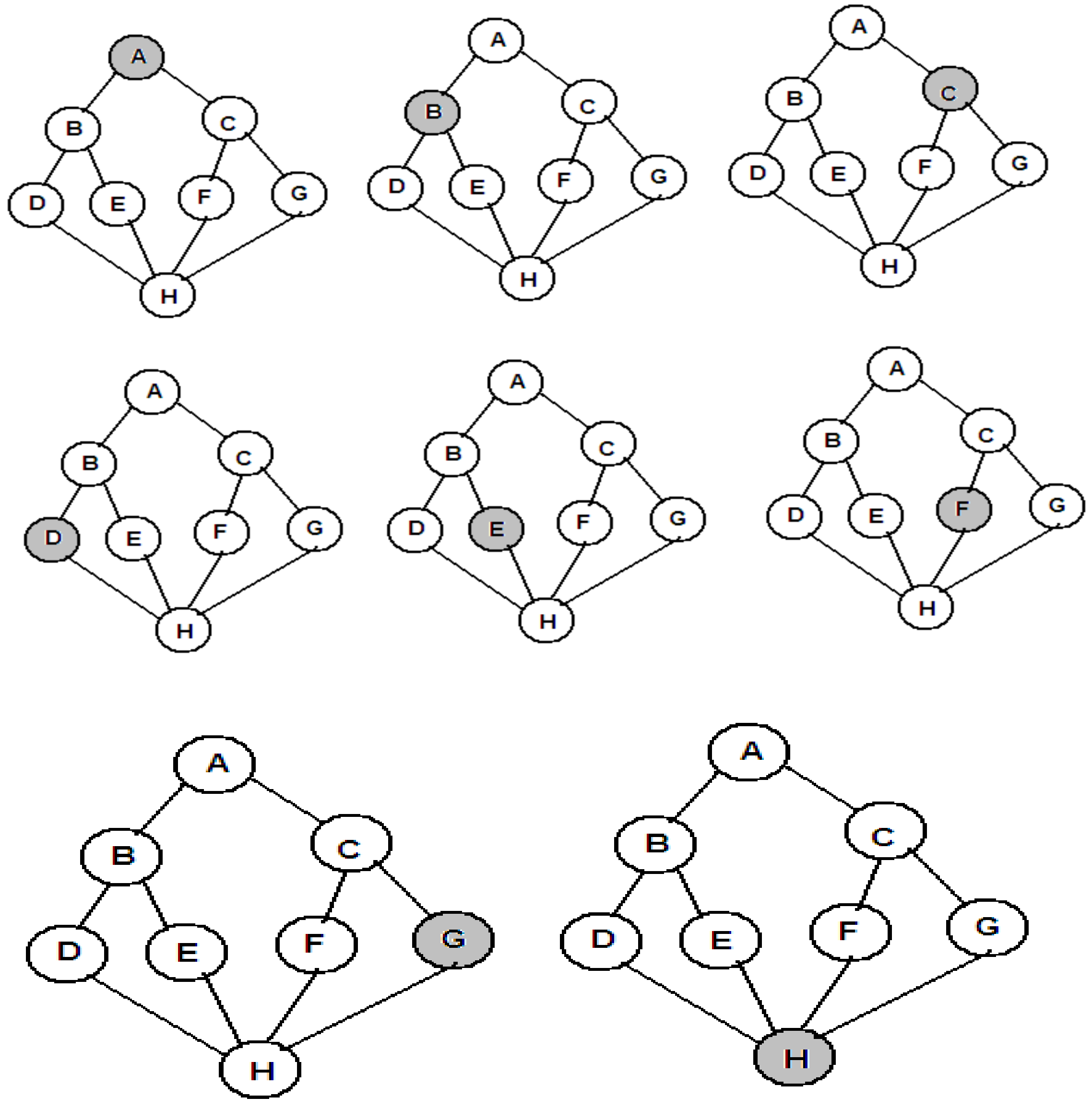
The Breadth First Traversal (BFT) algorithm calls the BFS algorithm on all the nodes.

Algorithm

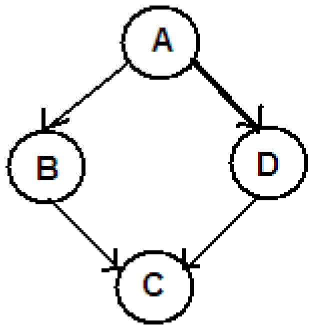
```
BFT(G, n)
Repeat for i = 1 to n
    Visited[i] = 0
End Repeat
Repeat for i = 1 to n
    If visited[i] = 0
        BFS(i)
    End if
End Repeat
```

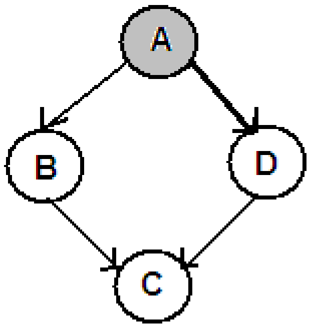
```
BFS(v)
u = v
visited[v] = 1
Repeat while(true)
    Repeat for all vertices w adjacent to u
        If visited[w] = 0
            Add w to queue
            Visited[w] = 1
        End if
    End Repeat
    If queue is empty
        Return
    End if
    Delete u from queue
End while
End BFS
```

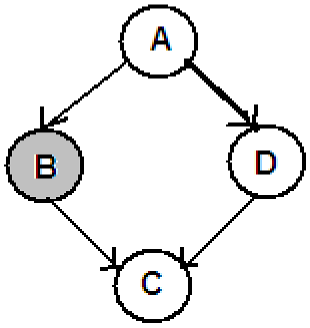


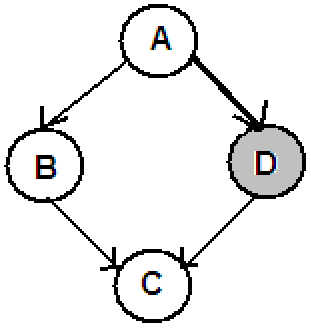


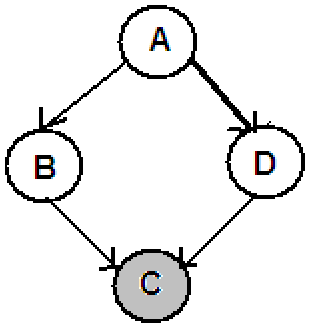
Now the following diagrams illustrate the BFS on a directed graph.











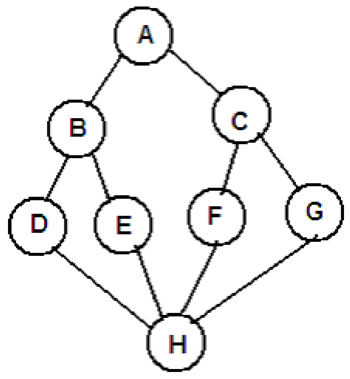
Depth First Search Traversal (DFS)

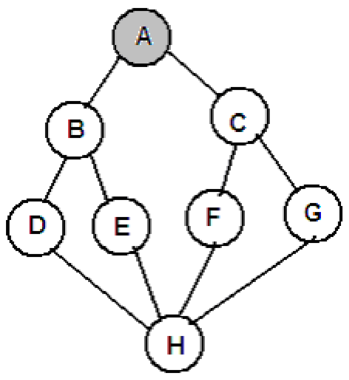
In the Depth First Search Traversal, as the name implies the nodes of the graph are traversed by searching through all the nodes by first going to the depth of the graph. The first node is visited first. Let this be node u. Find out all the adjacent nodes of u. Let that be w. Apply the DFS on the first adjacent node recursively. Since a recursive approach is followed, the nodes are traversed by going to the depth of the graph first. The DFT algorithm calls the DFS algorithm repeatedly for all the nodes in the graph.

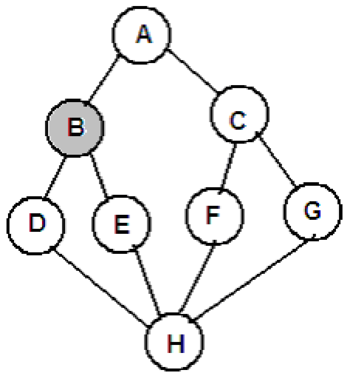
Algorithm

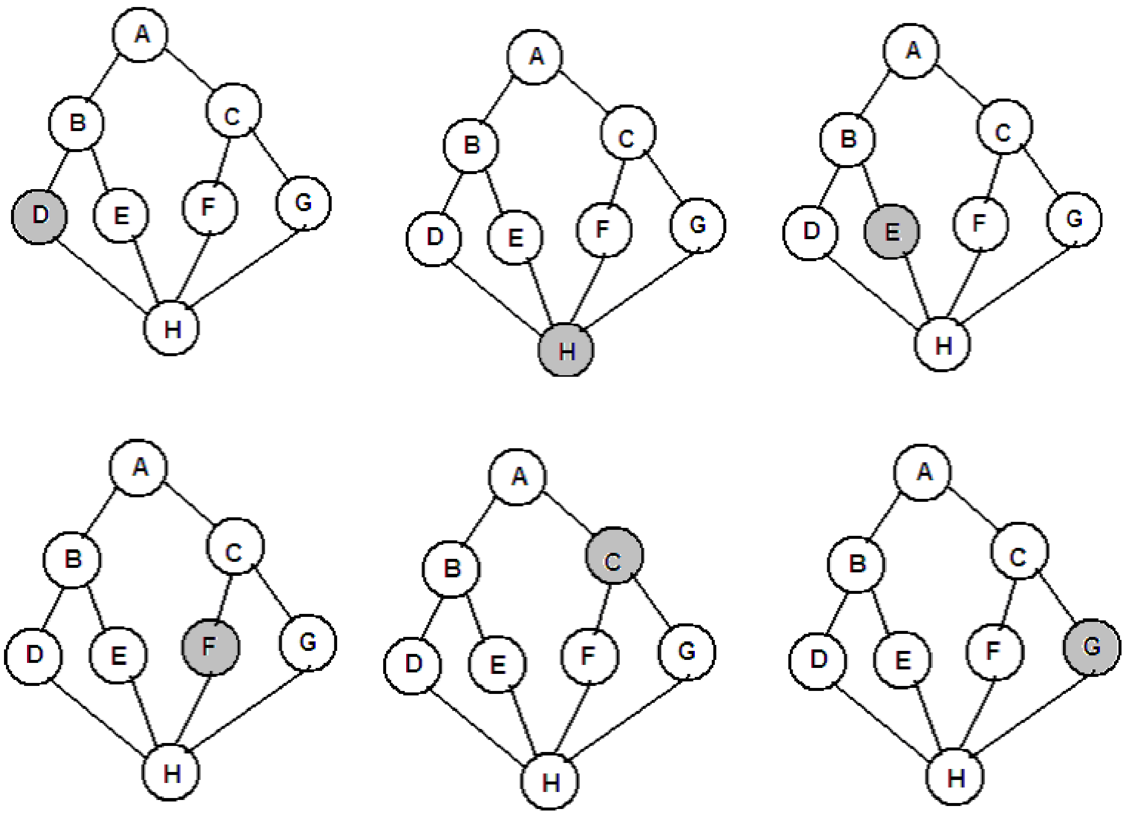
```
DFT(G, n)
Repeat for i = 1 to n
    Visited[i] = 0
End Repeat
Repeat for i = 1 to n
    If visited[i] = 0
        DFS(i)
    End if
End Repeat
```

```
DFS(v)
Visited[v] = 1
Repeat for each vertex w adjacent from v
    If visited[w] = 0
        DFS(w)
    End if
End for
```

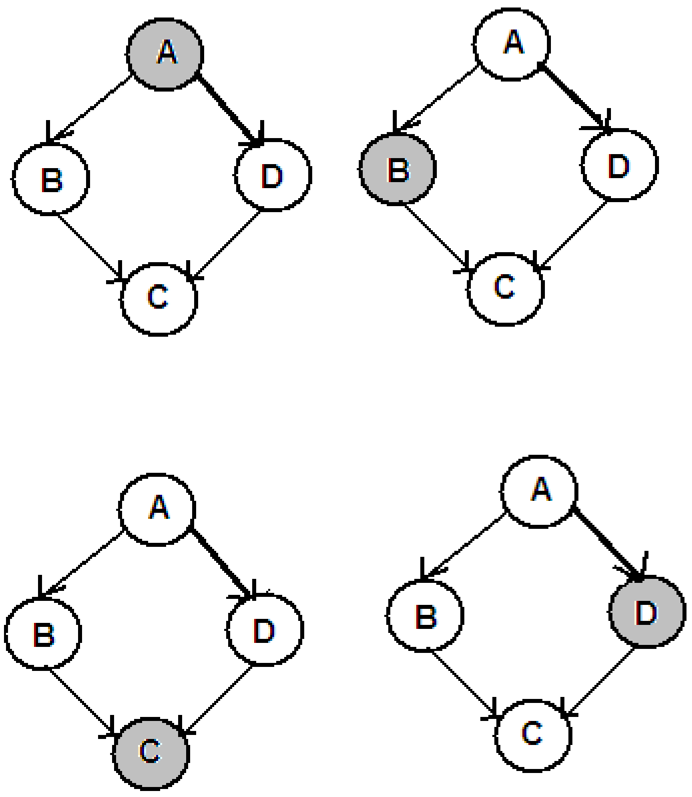








Now the following diagrams illustrate the DFS on a directed graph.



Applications of depth First Traversal

Depth-first search (DFS) is an algorithm (or technique) for traversing a graph.

1) For an unweighted graph, DFS traversal of the graph produces the minimum spanning tree and all pair shortest path tree.

2) Detecting cycle in a graph : A graph has cycle if and only if we see a back edge during DFS. So we can run DFS for the graph and check for back edges.

3) Path Finding:

We can specialize the DFS algorithm to find a path between two given vertices u and z.
i) Call DFS(G, u) with u as the start vertex.
ii) Use a stack S to keep track of the path between the start vertex and the current vertex.
iii) As soon as destination vertex z is encountered, return the path as the contents of the stack

4) Topological Sorting

Topological Sorting is mainly used for scheduling jobs from the given dependencies among jobs. In computer science, applications of this type arise in instruction scheduling, ordering of formula cell evaluation when recomputing formula values in spreadsheets, logic synthesis, determining the order of compilation tasks to perform in make files, data serialization, and resolving symbol dependencies in linkers .

5) **Finding** Strongly Connected Components **of a graph** A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex.

6) Solving puzzles with only one solution, such as mazes. DFS can be adapted to find all solutions to a maze by only including nodes on the current path in the visited set.

Applications of Breadth First Traversal

1) Shortest Path and Minimum Spanning Tree for unweighted graph In unweighted graph, the shortest path is the path with least number of edges. With Breadth First, we always reach a vertex from given source using minimum number of edges. Also, in case of unweighted graphs, any spanning tree is Minimum Spanning Tree and we can use either Depth or Breadth first traversal for finding a spanning tree.

2) Peer to Peer Networks. In Peer to Peer Networks like BitTorrent, Breadth First Search is used to find all neighbor nodes.

3) Crawlers in Search Engines: Crawlers build index using Breadth First. The idea is to start from source page and follow all links from source and keep doing same. Depth First Traversal can also be used for crawlers, but the advantage with Breadth First Traversal is, depth or levels of built tree can be limited.

4) Social Networking Websites: In social networks, we can find people within a given distance 'k' from a person using Breadth First Search till 'k' levels.

5) GPS Navigation systems: Breadth First Search is used to find all neighboring locations.

6) Broadcasting in Network: In networks, a broadcasted packet follows Breadth First Search to reach all nodes.

7) In Garbage Collection: Breadth First Search is used in copying garbage collection using Cheney's algorithm. Breadth First Search is preferred over Depth First Search because of better locality of reference:

8) Cycle detection in undirected graph: In undirected graphs, either Breadth First Search or Depth First Search can be used to detect cycle. In directed graph, only depth first search can be used.

9) Ford–Fulkerson algorithm In Ford-Fulkerson algorithm, we can either use Breadth First or Depth First Traversal to find the maximum flow. Breadth First Traversal is preferred as it reduces worst case time complexity to O(VE2).

10) To test if a graph is Bipartite We can either use Breadth First or Depth First Traversal.

11) Path Finding We can either use Breadth First or Depth First Traversal to find if there is a path between two vertices.

12) Finding all nodes within one connected component: We can either use Breadth First or Depth First Traversal to find all nodes reachable from a given node.

Many algorithms like Prim's Minimum Spanning Tree and Dijkstra's Single Source Shortest Path use structure similar to Breadth First Search.