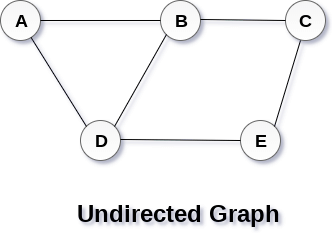
**Day 10**

**1) Graphs-**

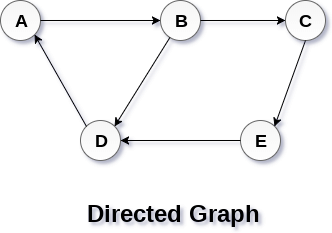
A graph can be defined as group of vertices/nodes and edges that are used to connect these vertices. A graph can be seen as a cyclic tree, where the vertices (Nodes) maintain any complex relationship among them instead of having parent child relationship.

**Undirected Graphs-**



In undirected graphs, if an edge exists between vertex A and B then the vertices can be traversed from B to A as well as A to B.

**Directed Graphs-**



In a directed graph, edges form an ordered pair. Edges represent a specific path from some vertex A to another vertex B. Node A is called initial node while node B is called terminal node. If an edge exists between vertex A and B and if points from A to B, then the vertices can be traversed from A to B only and not B to A.

**1.1 Graph Terminology-**

**Path**

A path can be defined as the sequence of nodes that are followed in order to reach some terminal node V from the initial node U.

**Closed Path**

A path will be called as closed path if the initial node is same as terminal node. A path will be closed path if V0=VN.

**Simple Path**

If all the nodes of the graph are distinct with an exception V0=VN, then such path P is called as closed simple path.

**Cycle**

A cycle can be defined as the path which has no repeated edges or vertices except the first and last vertices.

**Connected Graph**

A connected graph is the one in which some path exists between every two vertices (u, v) in V. There are no isolated nodes in connected graph.

**Complete Graph**

A complete graph is the one in which every node is connected with all other nodes. A complete graph contain n(n-1)/2 edges where n is the number of nodes in the graph.

**Weighted Graph**

In a weighted graph, each edge is assigned with some data such as length or weight. The weight of an edge e can be given as w(e) which must be a positive (+) value indicating the cost of traversing the edge.

**Digraph**

A digraph is a directed graph in which each edge of the graph is associated with some direction and the traversing can be done only in the specified direction.

**Loop**

An edge that is associated with the similar end points can be called as Loop.

**Adjacent Nodes**

If two nodes u and v are connected via an edge e, then the nodes u and v are called as neighbours or adjacent nodes.

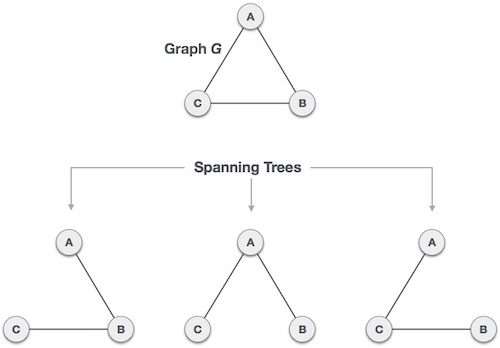
**Degree of the Node**

A degree of a node is the number of edges that are connected with that node. A node with degree 0 is called as isolated node.

**2) Spanning trees-**

A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected.

By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.



A complete undirected graph can have maximum n^(n-2) number of spanning trees, where n is the number of nodes. In the above addressed example, n is 3, hence 3^(3−2) = 3 spanning trees are possible.

**2.1) General Properties of Spanning Tree-**

We now understand that one graph can have more than one spanning tree. Following are a few properties of the spanning tree connected to graph G −

a) A connected graph G can have more than one spanning tree.

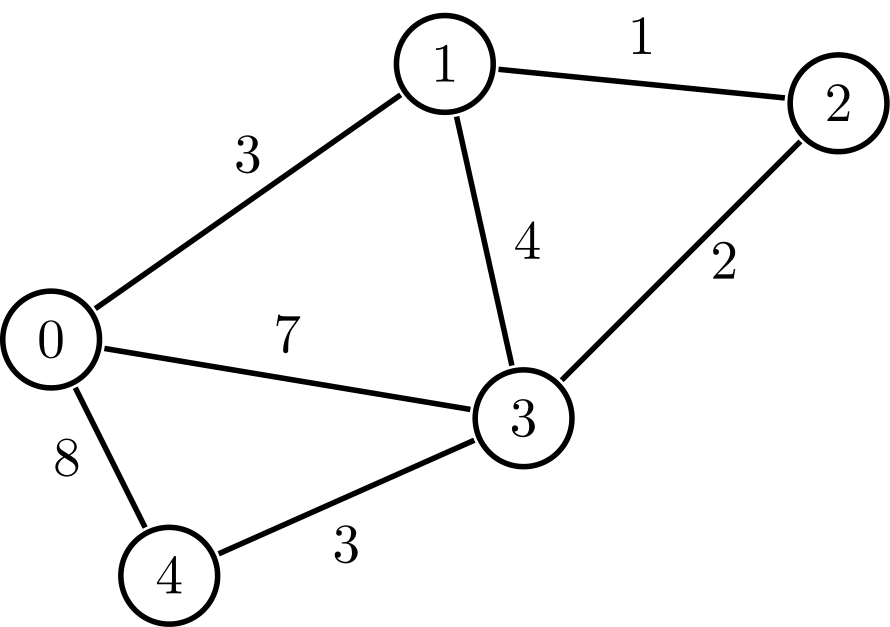
b) All possible spanning trees of graph G, have the same number of edges and vertices.

c) The spanning tree does not have any cycle (loops).

d) Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning tree is minimally connected.

e) Adding one edge to the spanning tree will create a circuit or loop, i.e. the spanning tree is maximally acyclic.

**2.2- Weighted graphs-**



A graph is a set of nodes and edges, where each edge represents a connection between two nodes. A weighted graph is a graph where each edge has a numerical value called weight. In real-world situations, this weight can be measured as distance, congestion, traffic load or any arbitrary value denoted to the edges.

**2.3) Minimum Spanning Tree (MST)-**

In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph.

**3) Two commonly used algorithms used for Graph traversal-**

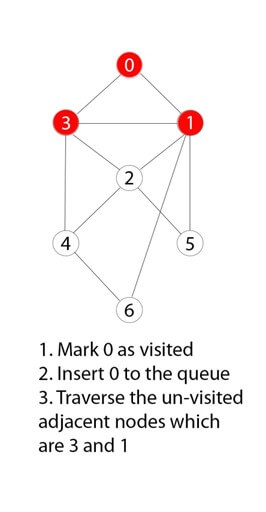
Traversing the graph means examining all the nodes and vertices of the graph.

**3.1 Breadth First Search (BFS) Algorithm-**

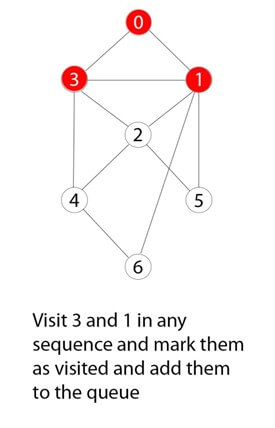
Breadth first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighbouring nodes. Then, it selects the nearest node and explore all the unexplored nodes. The algorithm follows the same process for each of the nearest node until it finds the goal or a leaf node is reached. The algorithm explores all neighbours of all the nodes and ensures that each node is visited exactly once and no node is visited twice. Commonly used data structure to implement BFS algorithm is queue.

Example-

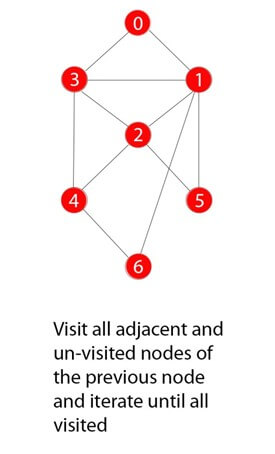
Step 1-



Step 2-



Step 3-

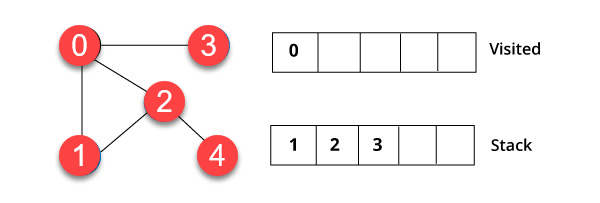


**3.2 Depth First Search (DFS) Algorithm-**

Depth first search (DFS) algorithm starts with the initial node of the graph G, and then goes to deeper and deeper until we find the goal node or the node which has no children. The algorithm, then backtracks from the dead-end towards the most recent node that is yet to be completely unexplored. The data structure which is being used in DFS is stack. In DFS, the edges that leads to an unvisited node are called discovery edges while the edges that leads to an already visited node are called block edges.

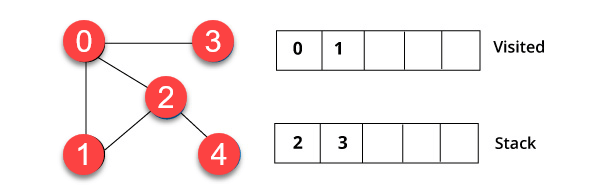
Example-

Step 1-



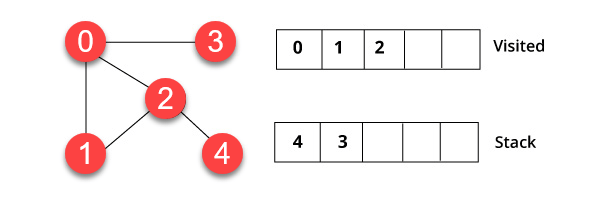
We have started from vertex 0. The algorithm begins by putting it in the visited list and simultaneously putting all its adjacent vertices in the data structure called stack.

Step 2-



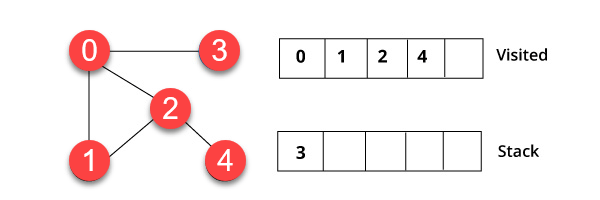
You will visit the element, which is at the top of the stack, for example, 1 and go to its adjacent nodes. It is because 0 has already been visited. Therefore, we visit vertex 2.

Step 3-

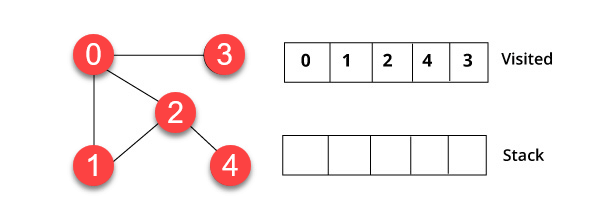


Vertex 2 has an unvisited nearby vertex in 4. Therefore, we add that in the stack and visit it.

Step 4-



Finally, we will visit the last vertex 3, it doesn't have any unvisited adjoining nodes. We have completed the traversal of the graph using DFS algorithm.



**3.3 Difference between BFS and DFS -**

**BFS DFS**

1) BFS finds the shortest path 1) DFS goes to the bottom

to the destination. of a subtree, then backtracks.

2) The full form of BFS is 2) The full form of DFS

Breadth-First Search. is Depth First Search.

3) It uses a queue to keep 3) It uses a stack to keep track

track of the next location of the next location to visit.

to visit.

4) BFS traverses according 4) DFS traverses according

to tree level. to tree depth.

5) It is implemented using 5) It is implemented using

FIFO list. LIFO list.

6) It requires more memory 6) It requires less memory as

as compare to DFS. compare to BFS.

7) This algorithm gives the 7) This algorithm doesn't

shallowest path solution. guarantee the shallowest

path solution.

8) There is no need of 8) There is a need of

backtracking in BFS. backtracking in DFS.

9) You can never be trapped 9) You can be trapped

into infinite loops. into infinite loops.

10) If you do not find any 10) If you do not find any goal

goal, you may need to the leaf node backtracking

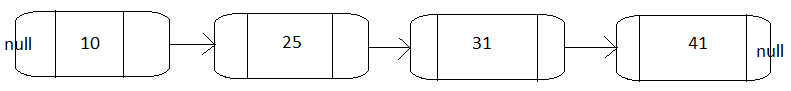
expand many nodes before may occur.

the solution is found.

**4) Types of linked lists-**

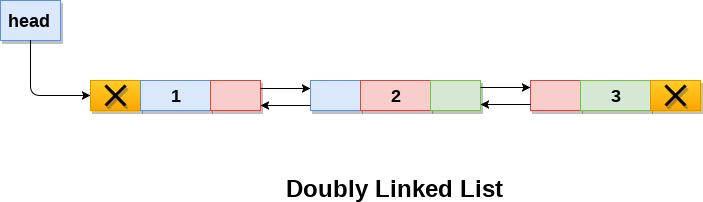
**4.1) Normal linked list -**

Linked list is a linear data structure which is used to maintain a list in the memory. It can be seen as the collection of nodes stored at non-contiguous memory locations. Each node of the list contains a pointer to its next node.



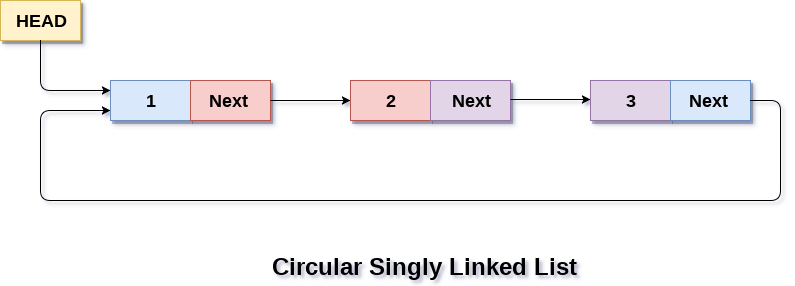
**4.2) Doubly linked list -**

Doubly linked list is a linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer).



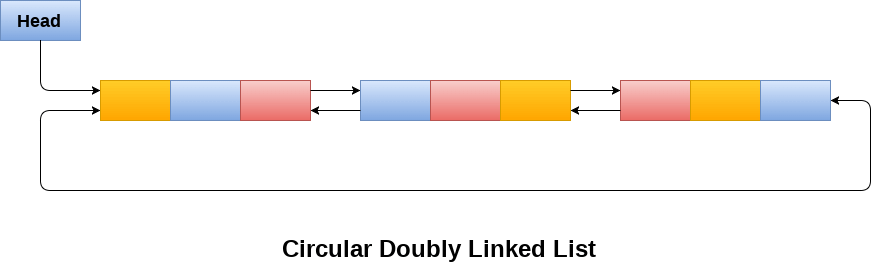
**4.3) Circular Singly linked list-**

In a circular Singly linked list, the last node of the list contains a pointer to the first node of the list. We can have circular singly linked list as well as circular doubly linked list.



**4.4) Circular Doubly linked list-**

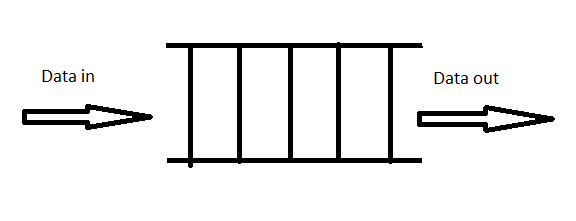
Circular doubly linked list contains pointers to its previous node as well as the next node. Circular doubly linked list doesn't contain NULL in any of the node. The last node of the list contains the address of the first node of the list. The first node of the list also contain address of the last node in its previous pointer.



**5) Queues-**

**5.1) Normal Queue-**

Queue is a linear list in which elements can be inserted only at one end called rear and deleted only at the other end called front.



**5.2) Circular Queue-**

In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we can not insert the next element even if there is a space in front of queue. In circular queue, last position is connected back to the first position to make a circle. It is also called ‘Ring Buffer’. Like normal queue, circular queue also works on FIFO (First In First Out) principle.

