Graphs

Introduction -

A graph is an abstract data structure that is used to implement the mathematical concept of graphs. It is basically a collection of vertices (also called nodes) and edges that connect these vertices. A graph is often viewed as a generalization of the tree structure, where instead of having a purely parent-to-child relationship between tree nodes, any kind of complex relationship can exist.

Definition -

A graph G is defined as an ordered set (V, E), where V(G) represents the set of vertices and E(G) represents the edges that connect these vertices.
A graph can be directed or undirected. In an undirected graph, edges do not have any direction associated with them. In a directed graph, edges form an ordered pair.

Graph Terminology -

Adjacent nodes or neighbours: For every edge, e = (u, v) that connects nodes u and v, the nodes u and v are the end-points and are said to be the adjacent nodes or neighbours.
Degree of a node: Degree of a node u, deg(u), is the total number of edges containing the node u. If deg(u) = 0, it means that u does not belong to any edge and such a node is known as an isolated node.
Regular graph: It is a graph where each vertex has the same number of neighbours. That is, every node has the same degree. A regular graph with vertices of degree k is called a k–regular graph or a regular graph of degree k.
Path: A path P written as P = {v0, v1, v2, ..., vn), of length n from a node u to v is defined as a sequence of (n+1) nodes. Here, u = v0, v = vn and vi–1 is adjacent to vifor i = 1, 2, 3,..., n.
Closed path: A path P is known as a closed path if the edge has the same end-points. That is, if v0 = vn.
Simple path: A path P is known as a simple path if all the nodes in the path are distinct with an
exception that v0 may be equal to vn. If v0 = vn, then the path is called a closed simple path.
Cycle: A path in which the first and the last vertices are the same. A simple cycle has no repeated edges or vertices (except the first and last vertices).

Connected graph: A graph is said to be connected if for any two vertices (u, v) in V there is a
path from u to v. That is to say that there are no isolated nodes in a connected graph. A connected graph that does not have any cycle is called a tree. Therefore, a tree is treated as a special graph.
Complete graph: A graph G is said to be complete if all its nodes are fully connected. That is,
there is a path from one node to every other node in the graph. A complete graph has n(n–1)/2 edges, where n is the number of nodes in G.
Clique: In an undirected graph G = (V, E), clique is a subset of the vertex set C Õ V, such that for every two vertices in C, there is an edge that connects two vertices.
Labelled graph or weighted graph: A graph is said to be labelled if every edge in the graph
is assigned some data. In a weighted graph, the edges of the graph are assigned some weight or length. The weight of an edge denoted by w(e) is a positive value which indicates the cost of
traversing the edge.
Multiple edges: Distinct edges which connect the same end-points are called multiple edges.
That is, e = (u, v) and e' = (u, v) are known as multiple edges of G.
Loop: An edge that has identical end-points is called a loop. That is, e = (u, u).
Multi-graph: A graph with multiple edges and/or loops is called a multi-graph. Figure 13.4(a)
shows a multi-graph.
Size of a graph: The size of a graph is the total number of edges in it.

Directed Graph -

A directed graph G, also known as a digraph, is a graph in which every edge has a direction assigned to it. An edge of a directed graph is given as an ordered pair (u, v) of nodes in G. For an edge (u, v),

- The edge begins at u and terminates at v.
- u is known as the origin or initial point of e. Correspondingly, v is known as the destination or terminal point of e.
- u is the predecessor of v. Correspondingly, v is the successor of u.
- Nodes u and v are adjacent to each other.

Terminology of a Directed Graph -

Out-degree of a node The out-degree of a node u, written as outdeg(u), is the number of edges that originate at u.

In-degree of a node The in-degree of a node u, written as indeg(u), is the number of edges that terminate at u.
Degree of a node The degree of a node, written as deg(u), is equal to the sum of in-degree and
out-degree of that node. Therefore, deg(u) = indeg(u) + outdeg(u).
Isolated vertex A vertex with degree zero. Such a vertex is not an end-point of any edge.
Pendant vertex (also known as leaf vertex) A vertex with degree one.
Cut vertex A vertex which when deleted would disconnect the remaining graph.
Source A node u is known as a source if it has a positive out-degree but a zero in-degree.
Sink A node u is known as a sink if it has a positive in-degree but a zero out-degree.
Reachability A node v is said to be reachable from node u, if and only if there exists a (directed) path from node u to node v.
Strongly connected directed graph A digraph is said to be strongly connected if and only if there exists a path between every pair of nodes in G. That is, if there is a path from node u to v, then there must be a path from node v to u.
Unilaterally connected graph A digraph is said to be unilaterally connected if there exists a path between any pair of nodes u, v in G such that there is a path from u to v or a path from v to u, but not both.
Weakly connected digraph A directed graph is said to be weakly connected if it is connected
by ignoring the direction of edges. That is, in such a graph, it is possible to reach any node from
any other node by traversing edges in any direction (may not be in the direction they point). The
nodes in a weakly connected directed graph must have either out-degree or in-degree of at least 1.
Parallel/Multiple edges Distinct edges which connect the same end-points are called multiple
edges. That is, e = (u, v) and e' = (u, v) are known as multiple edges of G.
Simple directed graph A directed graph G is said to be a simple directed graph if and only if it
has no parallel edges. However, a simple directed graph may contain cycles with an exception that it cannot have more than one loop at a given node.

Transitive Closure of a Directed Graph -

For a directed graph G = (V,E), where V is the set of vertices and E is the set of edges, the transitive closure of G is a graph G* = (V,E*). In G*, for every vertex pair v, w in V there is an edge (v, w) in E* if and only if there is a valid path from v to w in G.

ALGORITHM -

```
Transitive_Closure(A, t, n)
Step 1: SET i=1, j=1, k=1
Step 2: Repeat Steps 3 and 4 while i<=n
Step 3:         Repeat Step 4 while j<=n
Step 4:                 IF (A[i][j]=1)
                                SET t[i][j]=1
                        ELSE
                                SET t[i][j] =0
                INCREMENT j
                [END OF LOOP]
        INCREMENT i
        [END OF LOOP]
Step 5: Repeat Steps 6 to 11 while k<=n
Step 6:         Repeat Steps 7 to 1 while i<=n
Step 7:                 Repeat Steps 8 and 9 while j<=n
Step 8:                         SET t[i,j] = t[i][j] V (t[i][k] t[k][j])
Step 9:                         INCREMENT j
                        [END OF LOOP]
Step 10:            INCREMENT i
                [END OF LOOP]
Step 11:     INCREMENT k
          [END OF LOOP]
Step 12: END
```

Representation Of Graphs -

There are three common ways of storing graphs in the computer's memory. They are:

- Sequential representation by using an adjacency matrix.
- Linked representation by using an adjacency list that stores the neighbours of a node using a linked list.
- Adjacency multi-list which is an extension of linked representation.

Graph Traversal Algorithms -

By traversing a graph, we mean the method of examining the nodes and edges of the graph. There are two standard methods of graph traversal:

- Breadth-first search
- Depth-first search

While breadth-first search uses a queue as an auxiliary data structure to store nodes for further
processing, the depth-first search scheme uses a stack. But both these algorithms make use of
a variable STATUS. During the execution of the algorithm, every node in the graph will have the
variable STATUS set to 1 or 2, depending on its current state.

Status	State of the node	Description
1	Ready	The initial state of the node N
2	Waiting	Node N is placed on the queue or stack and waiting to be processed
3	Processed	Node N has been completely processed

Breadth-First Search Algorithm -

Breadth-first search (BFS) is a graph search algorithm that begins at the root node and explores all the neighbouring nodes. Then for each of those nearest nodes, the algorithm explores
their unexplored neighbour nodes, and so on, until it finds the goal.
We start examining the node A and then all the neighbours of A are examined. In the next step, we examine the neighbours of neighbours of A, so on and so forth. We need to track the neighbours of the node and guarantee that every node in the graph is processed and no node is processed more than once. This is accomplished by using a queue that will hold the nodes that are waiting for further processing and a variable STATUS to represent the current state of the node.

ALGORITHM -

```
Step 1: SET STATUS=1 (ready state) for each node in G
Step 2: Enqueue the starting node A and set its STATUS=2 (waiting state)
Step 3: Repeat Steps 4 and 5 until QUEUE is empty
Step 4: Dequeue a node N. Process it and set its STATUS=3 (processed state).
Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS=1) and set
        their STATUS=2 (waiting state)
        [END OF LOOP]
Step 6: EXIT
```

Applications of Breadth-First Search Algorithm -

Breadth-first search can be used to solve many problems such as:

- Finding all connected components in a graph G.
- Finding all nodes within an individual connected component.
- Finding the shortest path between two nodes, u and v, of an unweighted graph.
- Finding the shortest path between two nodes, u and v, of a weighted graph.

Depth-first Search Algorithm -

The depth-first search algorithm progresses by expanding the starting node of G and then going deeper and deeper until the goal node is found, or until a node that has no children is encountered. When a dead-end is reached, the algorithm backtracks, returning to the most recent node that has not been completely explored.
Depth-first search begins at a starting node A which becomes the current node. Then, it examines each node N along a path P which begins at A. That is, we process a neighbour of A, then a neighbour of A, and so on. During the execution of the algorithm, if we reach a path that has a node N that has already been processed, then we backtrack to the current node. Otherwise, the unvisited (unprocessed) node becomes the current node.
The algorithm proceeds like this until we reach a dead-end (end of path P). On reaching the dead-end, we backtrack to find another path P¢. The algorithm terminates when backtracking leads back to the starting node A. In this algorithm, edges that lead to a new vertex are called discovery edges and edges that lead to an already visited vertex are called back edges.

ALGORITHM -

```
Step 1: SET STATUS=1 (ready state) for each node in G
Step 2: Push the starting nodeAon the stack and set its STATUS=2 (waiting state)
Step 3: Repeat Steps 4 and 5 until STACK is empty
Step 4:         Pop the top node N. Process it and set its nSTATUS=3 (processed state)
Step 5:         Push on the stack all the neighbours of N that are in the ready state
(whose
                STATUS=1) and set their STATUS=2 (waiting state)
        [END OF LOOP]
Step 6: EXIT
```

Applications of Depth-First Search Algorithm -

Depth-first search is useful for:

- Finding a path between two specified nodes, u and v, of an unweighted graph.
- Finding a path between two specified nodes, u and v, of a weighted graph.
- Finding whether a graph is connected or not.

- Computing the spanning tree of a connected graph.

Topological Sorting -

Topological sort of a directed acyclic graph (DAG) G is defined as a linear ordering of its nodes
in which each node comes before all nodes to which it has outbound edges. Every DAG has one or more topological sorts.
A topological sort of a DAG G is an ordering of the vertices of G such that if G contains an
edge (u, v), then u appears before v in the ordering. Note that topological sort is possible only on directed acyclic graphs that do not have any cycles. For a DAG that contains cycles, no linear ordering of its vertices is possible.
In simple words, a topological ordering of a DAG G is an ordering of its vertices such that any directed path in G traverses the vertices in increasing order.
Topological sorting is widely used in scheduling applications, jobs, or tasks.
The jobs that have to be completed are represented by nodes, and there is an edge from node u to v if job u must be completed before job v can be started. A topological sort of such a graph gives an order in which the given jobs must be performed.

ALGORITHM -

```
Step 1: Find the in-degree INDEG(N) of every node graph
Step 2: Enqueue all the nodes with a zero in-degree
Step 3: Repeat Steps 4 and 5 until the QUEUE is empty
Step 4:         Remove the front nodeNof the QUEUE by setting
                FRONT = FRONT+1
Step 5:         Repeat for each neighbourMof node N:
                        a) Delete the edge from N to M by setting
                                INDEG(M) = INDEG(M)-1
                        b) IF INDEG(M) = , then Enqueue M, that is,
                                addMto the rear of the queue
                [END OF INNER LOOP]
        [END OF LOOP]
Step 6: EXIT
```