BYJU'S
EXAM PREP

Vision 2024

A Course for GATE & PSUs

Computer Science Engineering

Algorithm

CHAPTER 6

Graph Algorithms

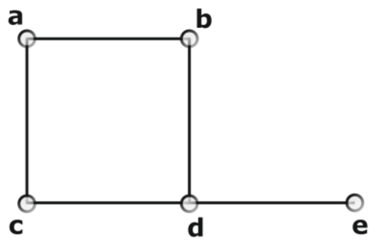
CHAPTER 6 ALGORITHM

GRAPH ALGORITHMS

1. GRAPHS

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as **vertices**, and the links that connect the vertices are called **edges**.

Formally, a graph is a pair of sets **(V, E)**, where **V** is the set of vertices and **E** is the set of edges, connecting the pairs of vertices. Take a look at the following graph –



In the above graph,

V = {a, b, c, d, e}

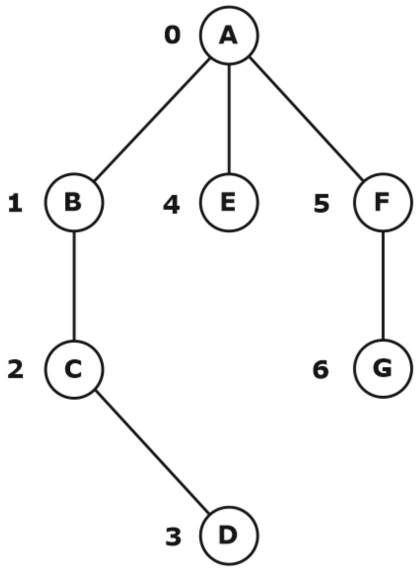
E = {ab, ac, bd, cd, de}

Graph Data Structure

Mathematical graphs can be represented in data structure. We can represent a graph using an array of vertices and a two-dimensional array of edges. Before we proceed further, let's familiarize ourselves with some important terms –

- **Vertex** – Each node of the graph is represented as a vertex. In the following example, the labeled circle represents vertices. Thus, A to G are vertices. We can represent them using an array as shown in the following image. Here A can be identified by index 0. B can be identified using index 1 and so on.
- **Edge** – Edge represents a path between two vertices or a line between two vertices. In the following example, the lines from A to B, B to C, and so on represent edges. We can use a two-dimensional array to represent an array as shown in the following image. Here AB can be represented as 1 at row 0, column 1, BC as 1 at row 1, column 2 and so on, keeping other combinations as 0.
- **Adjacency** – Two node or vertices are adjacent if they are connected to each other through an edge. In the following example, B is adjacent to A, C is adjacent to B, and so on.

- **Path** – Path represents a sequence of edges between the two vertices. In the following example, ABCD represents a path from A to D.



Basic Operations

Following are basic primary operations of a Graph –

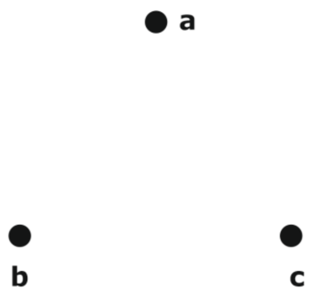
- **Add Vertex** – Adds a vertex to the graph.
- **Add Edge** – Adds an edge between the two vertices of the graph.
- **Display Vertex** – Displays a vertex of the graph.

2. Types of Graphs

2.1 Null Graph

A graph having no edges is called a Null Graph.

Example



In the above graph, there are three vertices named 'a', 'b', and 'c', but there are no edges among them. Hence it is a Null Graph.

2.2 Trivial Graph

A graph with only one vertex is called a Trivial Graph.

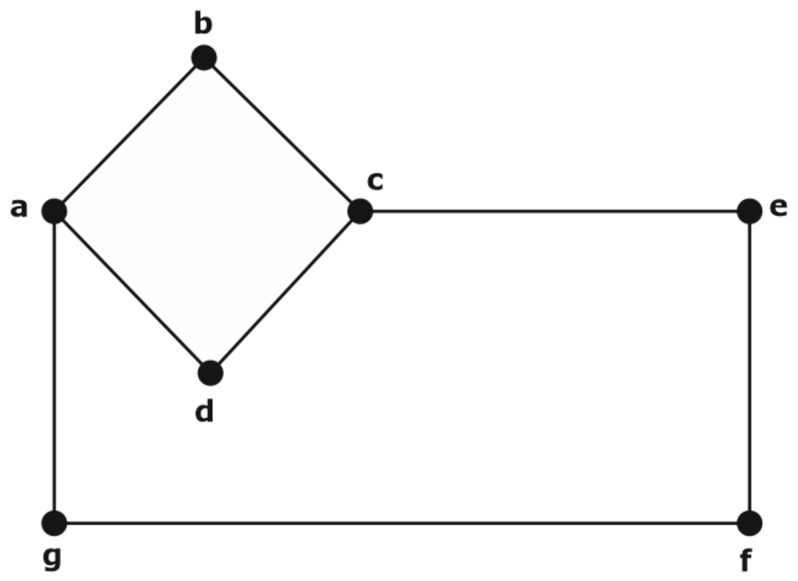
Example

In the above shown graph, there is only one vertex 'a' with no other edges. Hence it is a Trivial graph.

2.3 Non-Directed Graph

A non-directed graph contains edges but the edges are not directed ones.

Example

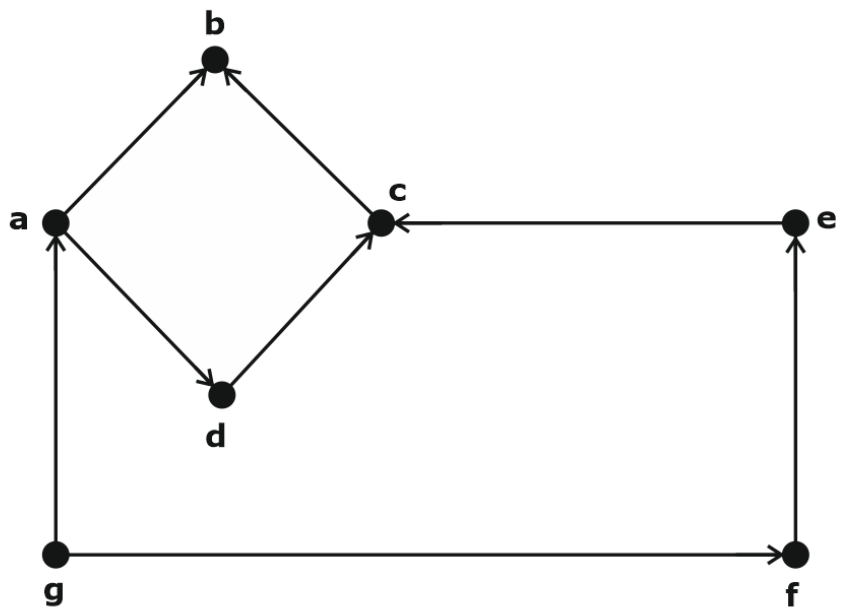


In this graph, ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’ are the vertices, and ‘ab’, ‘bc’, ‘cd’, ‘da’, ‘ag’, ‘gf’, ‘ef’ are the edges of the graph. Since it is a non-directed graph, the edges ‘ab’ and ‘ba’ are the same. Similarly other edges are also considered in the same way.

2.4 Directed Graph

In a directed graph, each edge has a direction.

Example



In the above graph, we have seven vertices ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, and ‘g’, and eight edges ‘ab’, ‘cb’, ‘dc’, ‘ad’, ‘ec’, ‘fe’, ‘gf’, and ‘ga’. As it is a directed graph, each edge bears an arrow mark that shows its direction. Note that in a directed graph, ‘ab’ is different from ‘ba’.

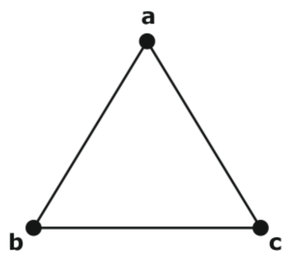
2.5 Simple Graph

A graph with no loops and no parallel edges is called a simple graph.

- The maximum number of edges possible in a single graph with 'n' vertices is ${}^nC_2$ where ${}^nC_2 = n(n - 1)/2$.
- The number of simple graphs possible with 'n' vertices $= 2^nc_2 = 2^{n(n-1)/2}$.

Example

In the following graph, there are 3 vertices with 3 edges which is maximum excluding the parallel edges and loops. This can be proved by using the above formulae.



The maximum number of edges with n=3 vertices –

${}^nC_2 = n(n-1)/2$

$= 3(3-1)/2$

$= 6/2$

$= 3$ edges

The maximum number of simple graphs with n=3 vertices –

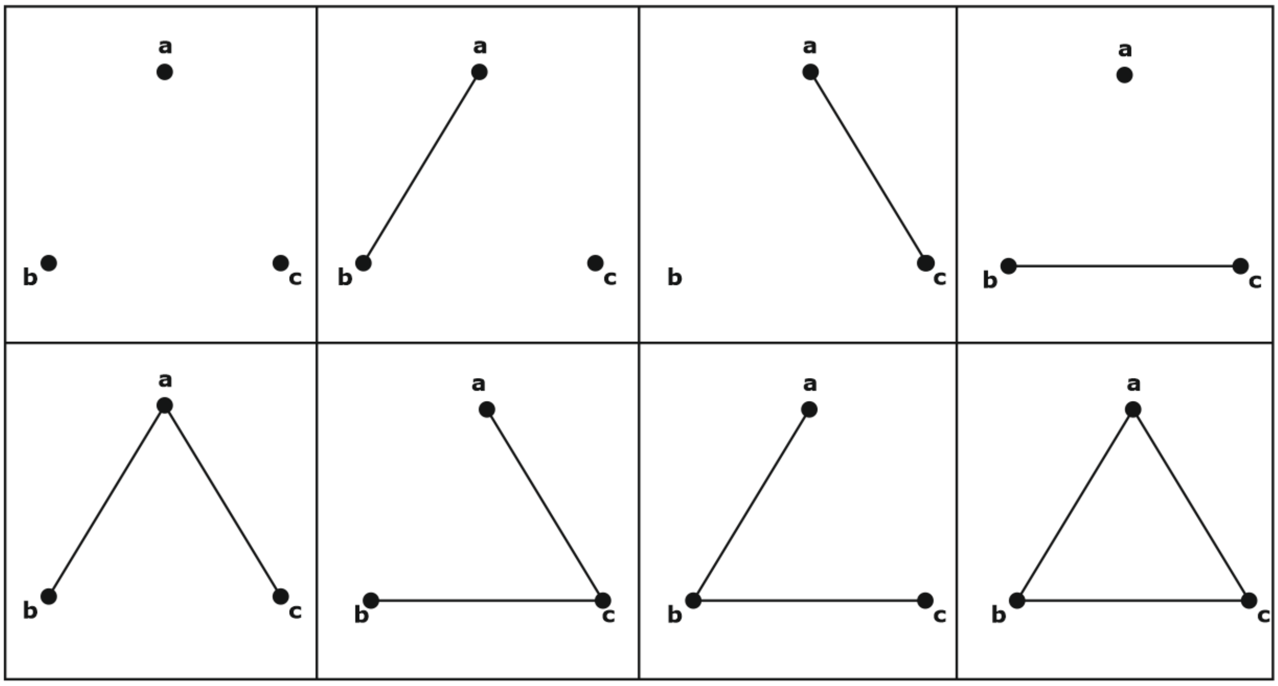
$2^nC_2 = 2^{n(n-1)/2}$

$= 2^{3(3-1)/2}$

$= 2^3$

$= 8$

These 8 graphs are as shown below –

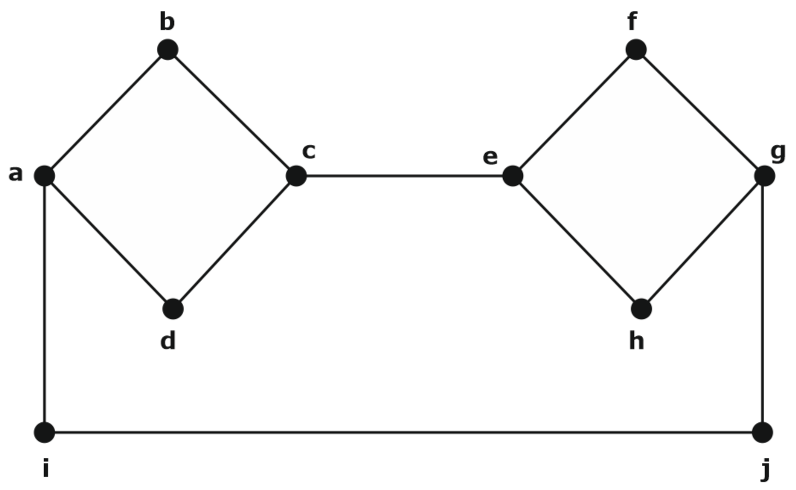


2.6 Connected Graph

A graph G is said to be connected **if there exists a path between every pair of vertices**. There should be at least one edge for every vertex in the graph. So that we can say that it is connected to some other vertex at the other side of the edge.

Example

In the following graph, each vertex has its own edge connected to another edge. Hence it is a connected graph.

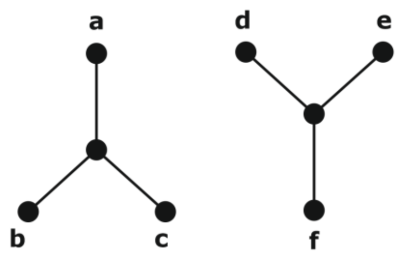


2.7 Disconnected Graph

A graph G is disconnected, if it does not contain at least two connected vertices.

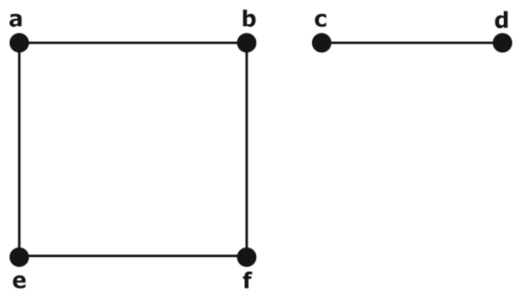
Example 1

The following graph is an example of a Disconnected Graph, where there are two components, one with ‘a’, ‘b’, ‘c’, ‘d’ vertices and another with ‘e’, ’f’, ‘g’, ‘h’ vertices.



The two components are independent and not connected to each other. Hence it is called disconnected graph.

Example 2



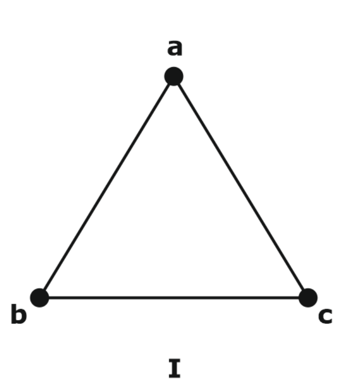
In this example, there are two independent components, a-b-f-e and c-d, which are not connected to each other. Hence this is a disconnected graph.

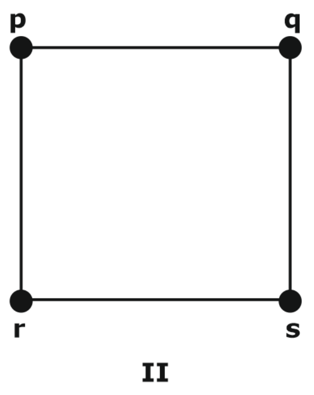
2.8 Regular Graph

A graph G is said to be regular, **if all its vertices have the same degree**. In a graph, if the degree of each vertex is ‘k’, then the graph is called a ‘k-regular graph’.

Example

In the following graphs, all the vertices have the same degree. So these graphs are called regular graphs.





In both the graphs, all the vertices have degree 2. They are called 2-Regular Graphs.

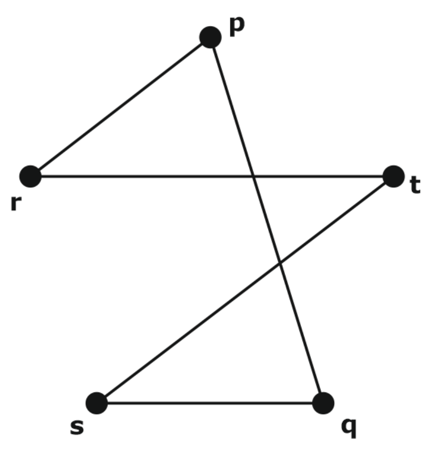
Complete Graph

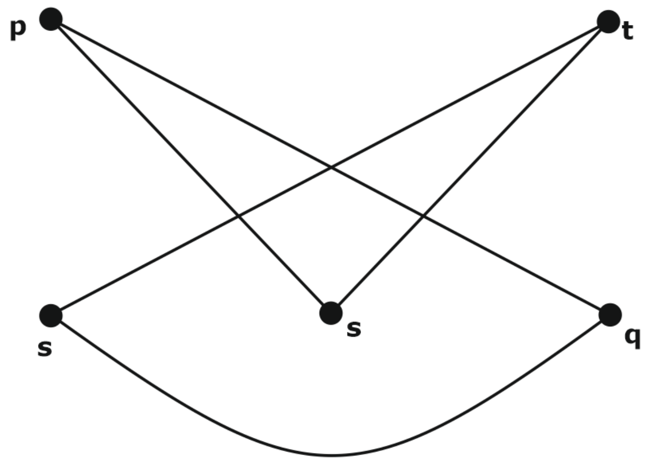
A simple graph with ‘n’ mutual vertices is called a complete graph and it is **denoted by ‘K_n’**. In the graph, **a vertex should have edges with all other vertices,** then it called a complete graph.

In other words, if a vertex is connected to all other vertices in a graph, then it is called a complete graph.

Example

In the following graphs, each vertex in the graph is connected with all the remaining vertices in the graph except by itself.





2.9 Cycle Graph

A simple graph with 'n' vertices (n >= 3) and 'n' edges is called a cycle graph if all its edges form a cycle of length 'n'.

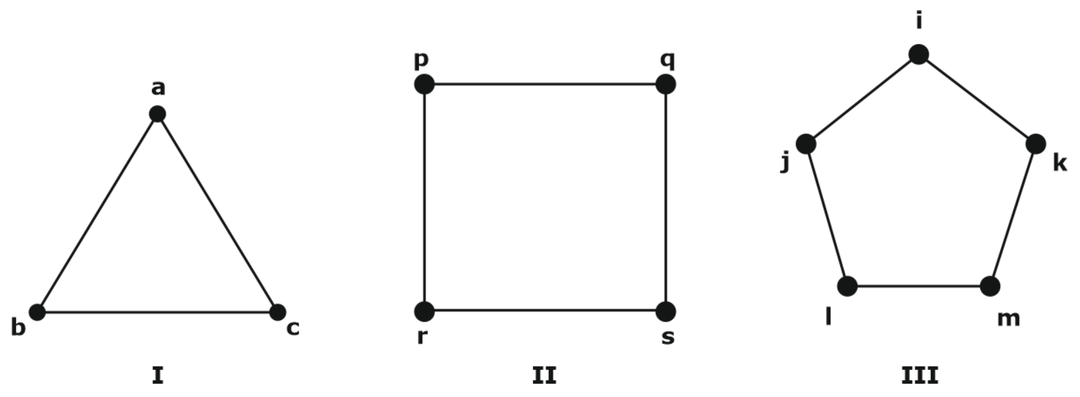
If the degree of each vertex in the graph is two, then it is called a Cycle Graph.

Notation – C_n

Example

Take a look at the following graphs –

- Graph I has 3 vertices with 3 edges which form a cycle 'ab-bc-ca'.
- Graph II has 4 vertices with 4 edges which form a cycle 'pq-qs-sr-rp'.
- Graph III has 5 vertices with 5 edges which form a cycle 'ik-km-ml-lj-ji'.



Hence all the given graphs are cycle graphs.

3. Graph Traversal

Graph traversal is a technique used for a searching vertex in a graph. The graph traversal is also used to decide the order of vertices visited in the search process. A graph traversal finds the edges to be used in the search process without creating loops. That means using graph traversal we visit all the vertices of the graph without getting into a looping path. There are two graph traversal techniques and they are as follows.

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

DFS (Depth First Search)

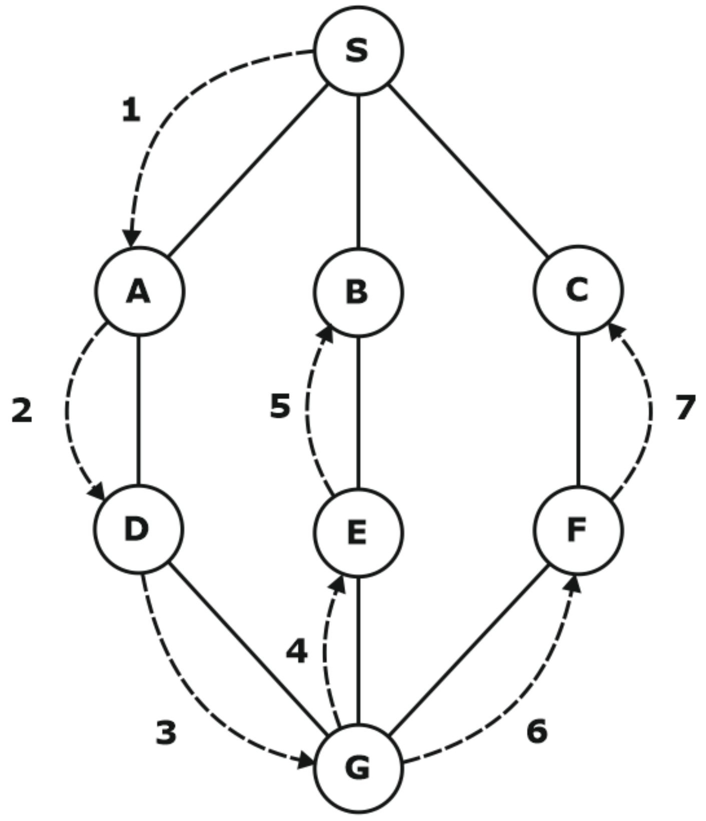
DFS traversal of a graph produces a spanning tree as final result. Spanning Tree is a graph without loops. We use Stack data structure with maximum size of total number of vertices in the graph to implement DFS traversal.

We use the following steps to implement DFS traversal...

- **Step 1 -** Define a Stack of size total number of vertices in the graph.
- **Step 2 -** Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.

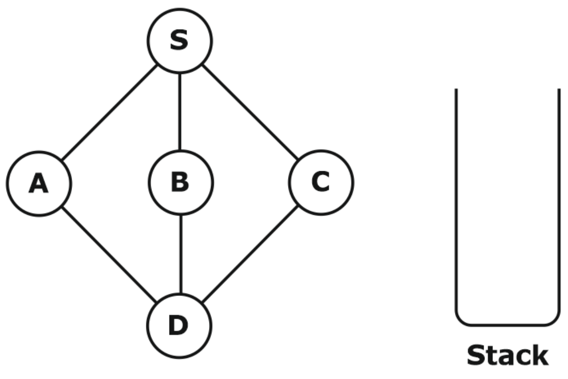
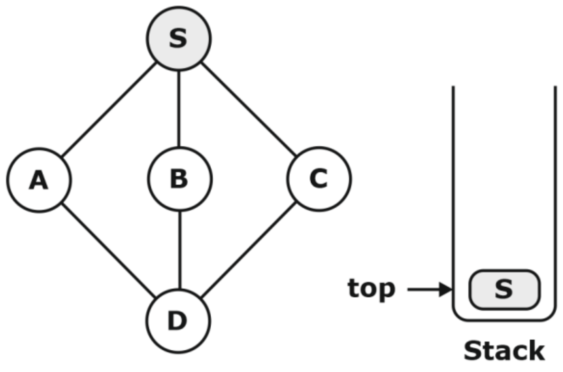
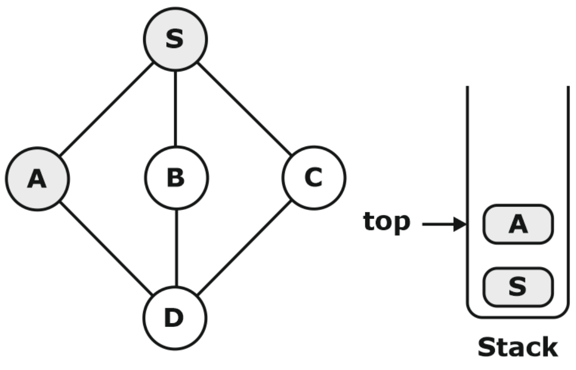
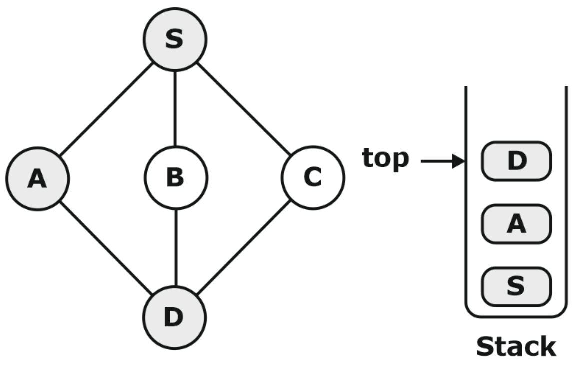
- **Step 3 -** Visit any one of the non-visited adjacent vertices of a vertex which is at the top of stack and push it onto the stack.
- **Step 4 -** Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.
- **Step 5 -** When there is no new vertex to visit then use backtracking and pop one vertex from the stack.
- **Step 6 -** Repeat steps 3, 4 and 5 until stack becomes Empty.
- **Step 7 -** When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph.

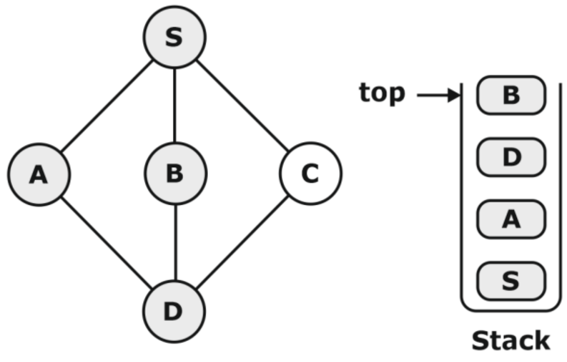
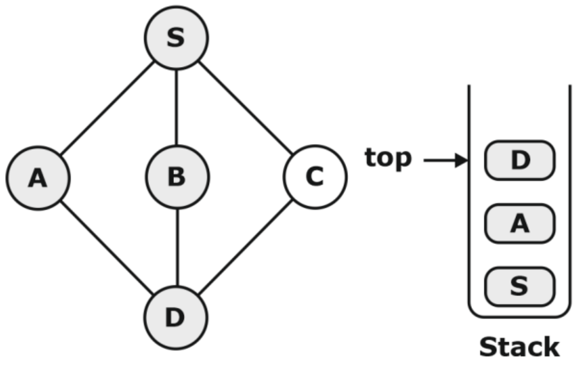
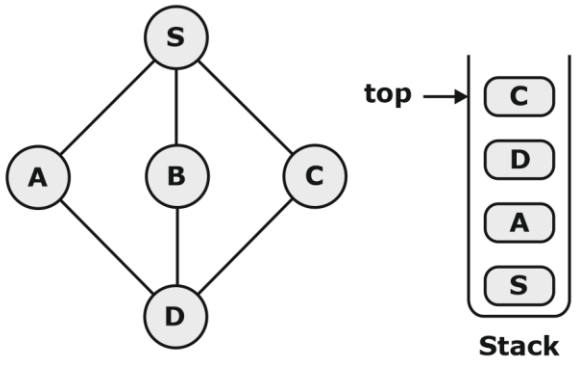
Depth First Search (DFS) algorithm traverses a graph in a depth ward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



As in the example given above, the DFS algorithm traverses from S to A to D to G to E to B first, then to F and lastly to C. It employs the following rules.

- **Rule 1** – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.
- **Rule 2** – If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
- **Rule 3** – Repeat Rule 1 and Rule 2 until the stack is empty.

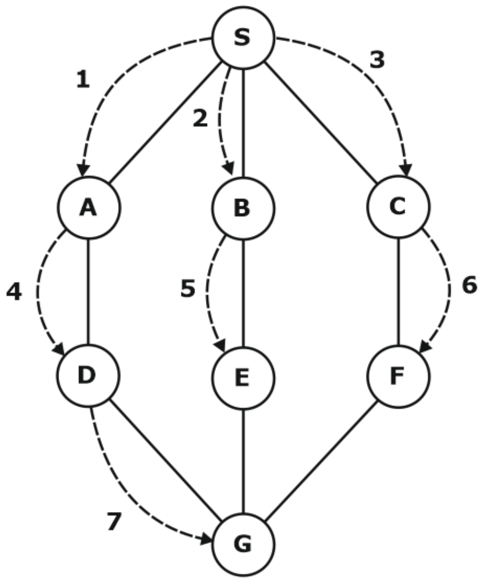
Step	Traversal	Description
1		Initialize the stack.
2		Mark **S** as visited and put it onto the stack. Explore any unvisited adjacent node from **S**. We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order.
3		Mark **A** as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both **S** and **D** are adjacent to **A** but we are concerned for unvisited nodes only.
4		Visit **D** and mark it as visited and put onto the stack. Here, we have **B** and **C** nodes, which are adjacent to **D** and both are unvisited. However, we shall again choose in an alphabetical order.

Step	Traversal	Description
5		We choose **B**, mark it as visited and put onto the stack. Here **B** does not have any unvisited adjacent node. So, we pop **B** from the stack.
6		We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find **D** to be on the top of the stack.
7		Only unvisited adjacent node is from **D** is **C** now. So we visit **C**, mark it as visited and put it onto the stack.

As **C** does not have any unvisited adjacent node so we keep popping the stack until we find a node that has an unvisited adjacent node. In this case, there's none and we keep popping until the stack is empty.

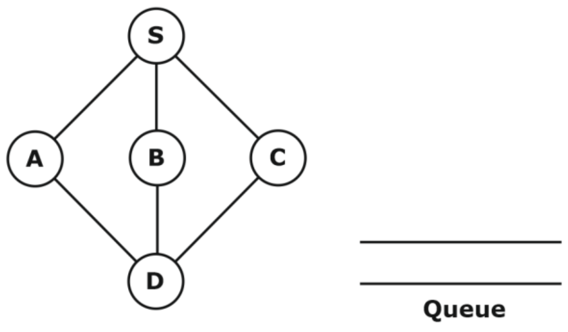
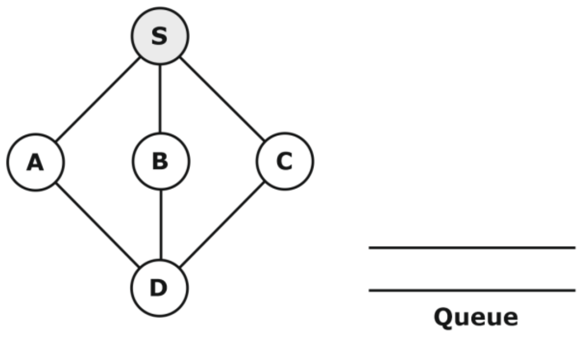
Breadth first Search

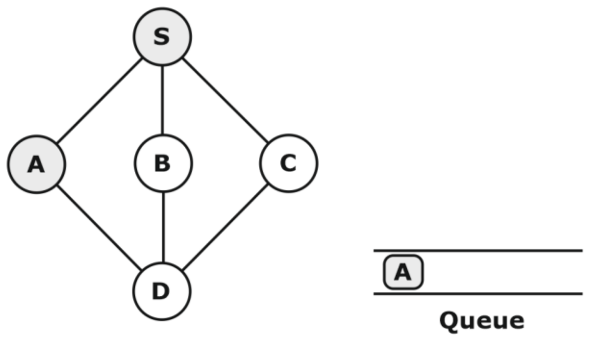
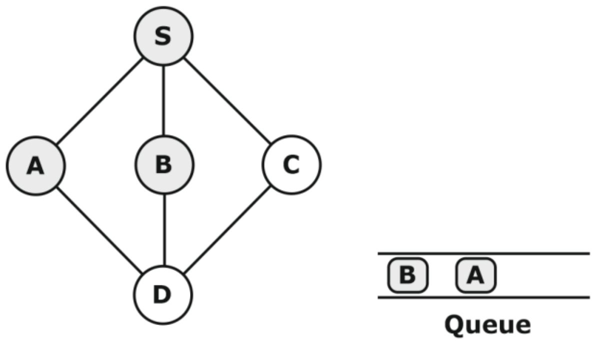
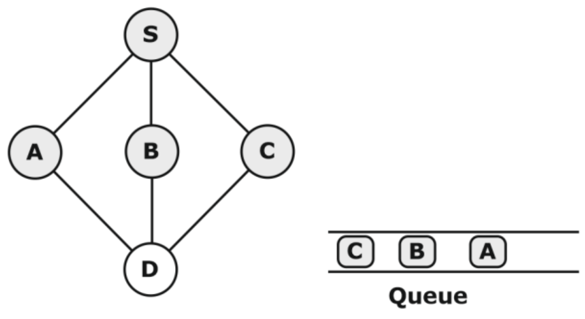
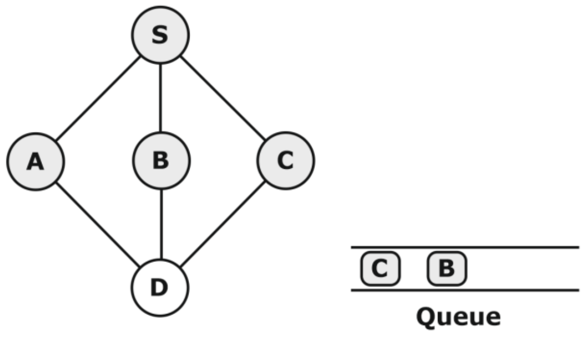
Breadth First Search (BFS) algorithm traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

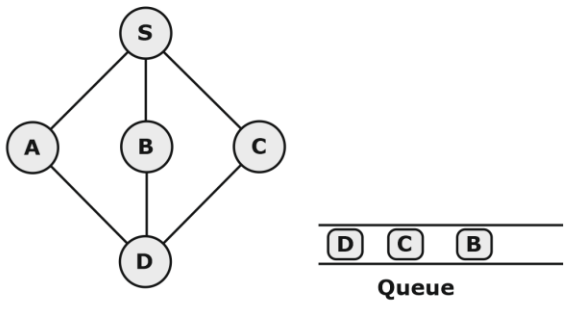


As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs the following rules.

- **Rule 1** – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
- **Rule 2** – If no adjacent vertex is found, remove the first vertex from the queue.
- **Rule 3** – Repeat Rule 1 and Rule 2 until the queue is empty.

Step	Traversal	Description
1		Initialize the queue.
2		We start from visiting **S** (starting node), and mark it as visited.

Step	Traversal	Description
3		We then see an unvisited adjacent node from **S**. In this example, we have three nodes but alphabetically we choose **A**, mark it as visited and enqueue it.
4		Next, the unvisited adjacent node from **S** is **B**. We mark it as visited and enqueue it.
5		Next, the unvisited adjacent node from **S** is **C**. We mark it as visited and enqueue it.
6		Now, **S** is left with no unvisited adjacent nodes. So, we dequeue and find **A**.

Step	Traversal	Description
7		From **A** we have **D** as unvisited adjacent node. We mark it as visited and enqueue it.

At this stage, we are left with no unmarked (unvisited) nodes. But as per the algorithm we keep on dequeuing in order to get all unvisited nodes. When the queue gets emptied, the program is over.

Difference between DFT and BFT

BFT	DFT
BFS stands for Breadth First Search.	DFS stands for Depth First Search.
BFS(Breadth First Search) uses Queue data structure for finding the shortest path.	DFS(Depth First Search) uses Stack data structure.
BFS can be used to find a single source shortest path in an unweighted graph, because in BFS, we reach a vertex with minimum number of edges from a source vertex.	In DFS, we might traverse through more edges to reach a destination vertex from a source.
BFS is more suitable for searching vertices which are closer to the given source.	DFS is more suitable when there are solutions away from source.
BFS considers all neighbours first and therefore not suitable for decision making trees used in games or puzzles.	DFS is more suitable for game or puzzle problems. We make a decision, then explore all paths through this decision. And if this decision leads to win situation, we stop.
The Time complexity of BFS is O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used, where V stands for vertices and E stands for edges.	The Time complexity of DFS is also O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used, where V stands for vertices and E stands for edges.

****