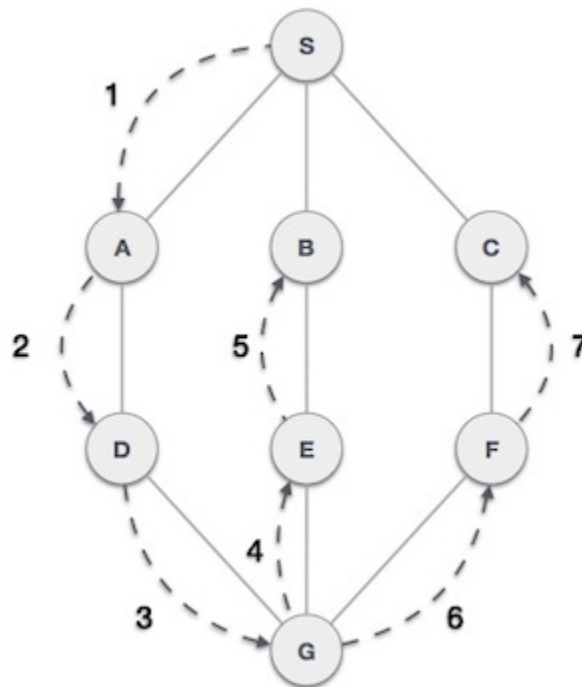


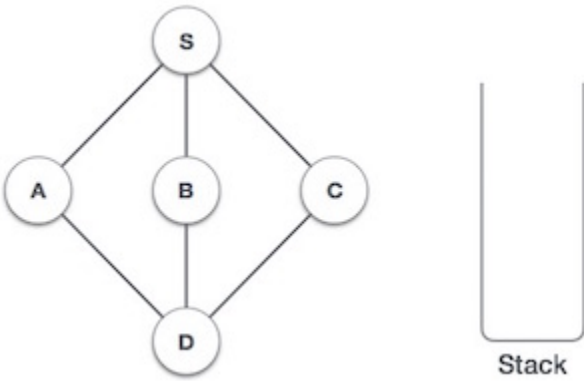
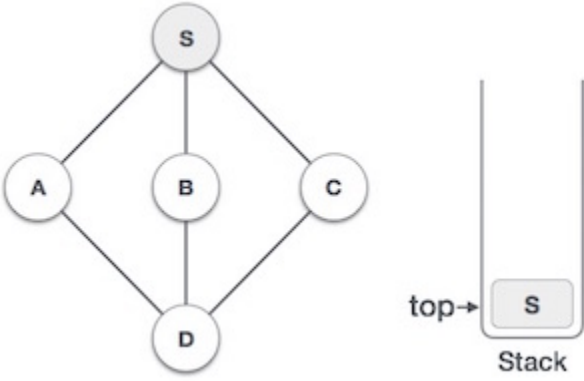
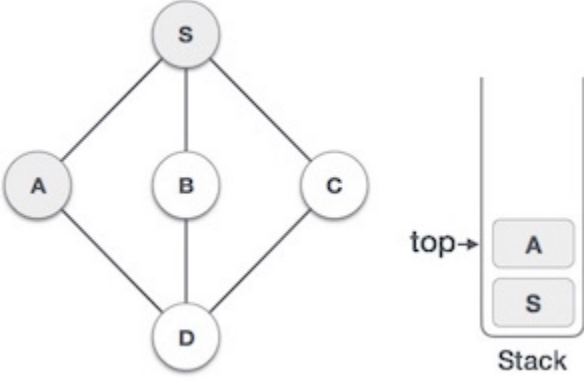
# Data Structure - Depth First Traversal

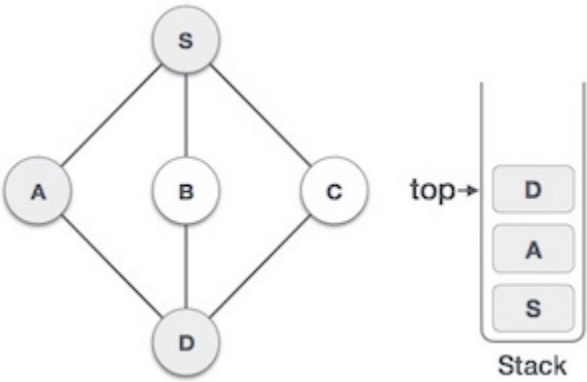
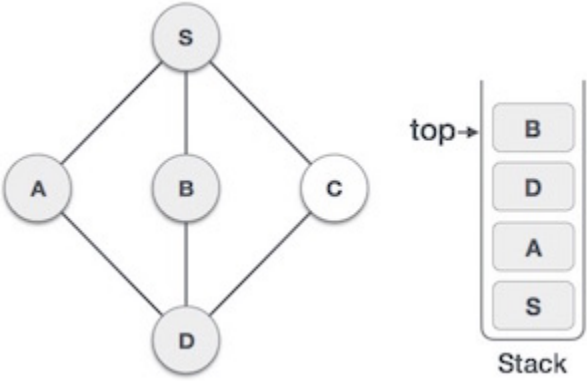
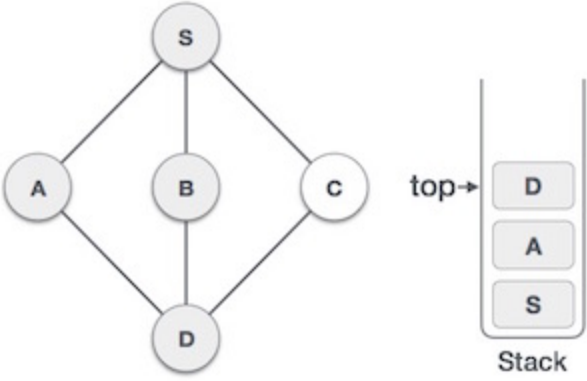
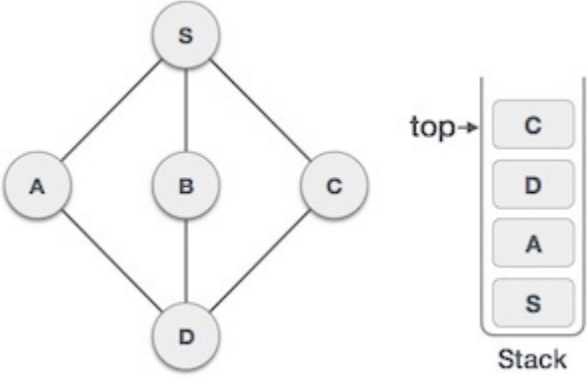
Depth First Search (DFS) algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



As in the example given above, 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 <b>S</b> as visited and put it onto the stack. Explore any unvisited adjacent node from <b>S</b> . 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 <b>A</b> as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both <b>S</b> and <b>D</b> are adjacent to <b>A</b> but we are concerned for unvisited nodes only.

4	 <p>The graph shows a diamond shape with nodes S at the top, A, B, and C in the middle, and D at the bottom. Edges connect S to A, B, and C; A to D; B to D; and C to D. The stack is a vertical container with nodes S, A, and D from bottom to top. A 'top' pointer points to the top node D.</p>	Visit <b>D</b> and mark it as visited and put onto the stack. Here, we have <b>B</b> and <b>C</b> nodes, which are adjacent to <b>D</b> and both are unvisited. However, we shall again choose in an alphabetical order.
5	 <p>The graph is the same as in step 4. The stack now contains nodes S, A, and B from bottom to top. The 'top' pointer points to the top node B.</p>	We choose <b>B</b> , mark it as visited and put onto the stack. Here <b>B</b> does not have any unvisited adjacent node. So, we pop <b>B</b> from the stack.
6	 <p>The graph is the same as in step 4. The stack now contains nodes S, A, and D from bottom to top. The 'top' pointer points to the top node D.</p>	We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find <b>D</b> to be on the top of the stack.
7	 <p>The graph is the same as in step 4. The stack now contains nodes S, A, D, and C from bottom to top. The 'top' pointer points to the top node C.</p>	Only unvisited adjacent node is from <b>D</b> is <b>C</b> now. So we visit <b>C</b> , 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.

To know about the implementation of this algorithm in C programming language, click [here](#) .