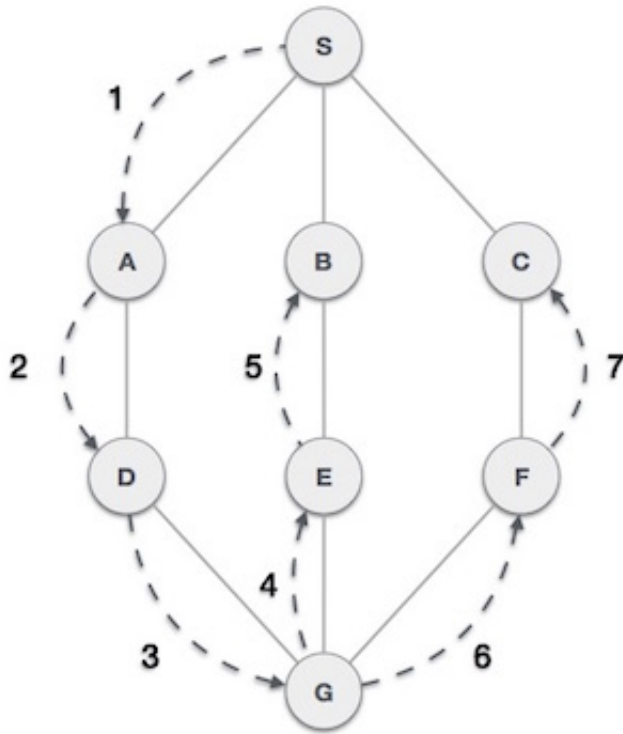


Data Structure - Depth First Traversal

https://www.tutorialspoint.com/data_structures_algorithms/depth_first_traversal.htm

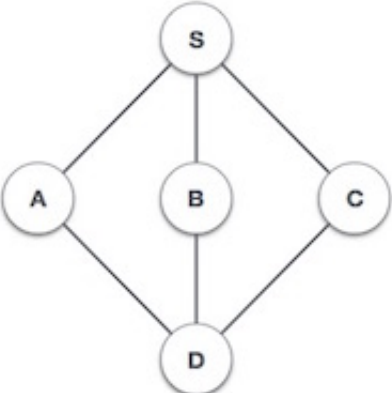

Copyright © tutorialspoint.com

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.

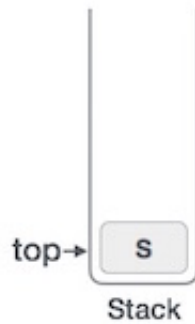
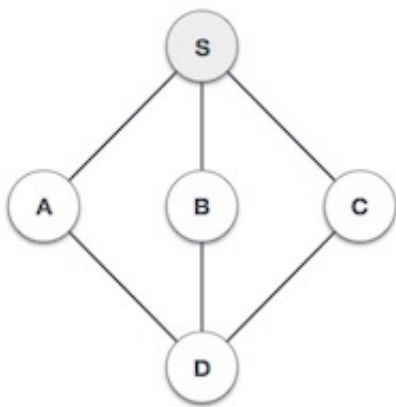


As in the example given above, DFS algorithm traverses from A to B to C to D first then to E, then to F and lastly to G. It employs the following rules.

- **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.

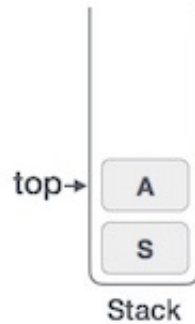
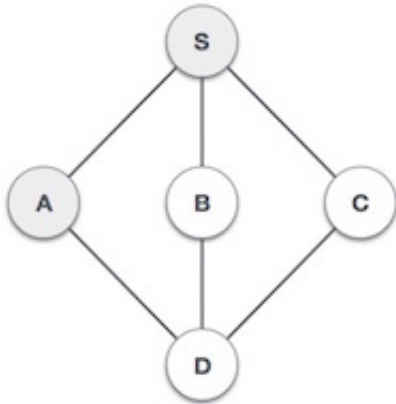
Step	Traversal	Description
1.		<div> Stack</div> <div>Initialize the stack.</div>

2.



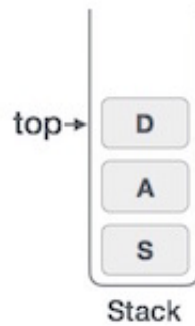
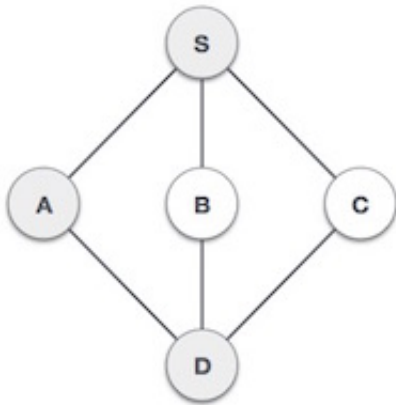
Mark **S** as visited and put it onto the stack. Explore any unvisited adjacent node from **S**. We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order.

3.



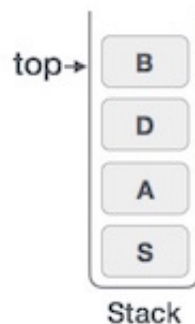
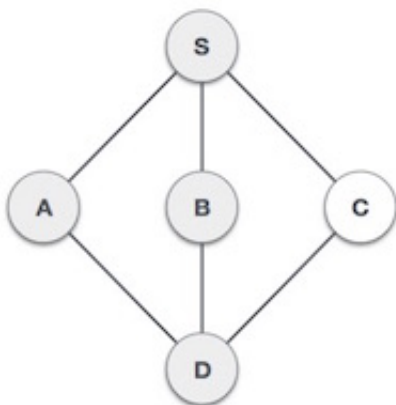
Mark **A** as visited and put it onto the stack. Explore any unvisited adjacent node from **A**. Both **S** and **D** are adjacent to **A** but we are concerned for unvisited nodes only.

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.

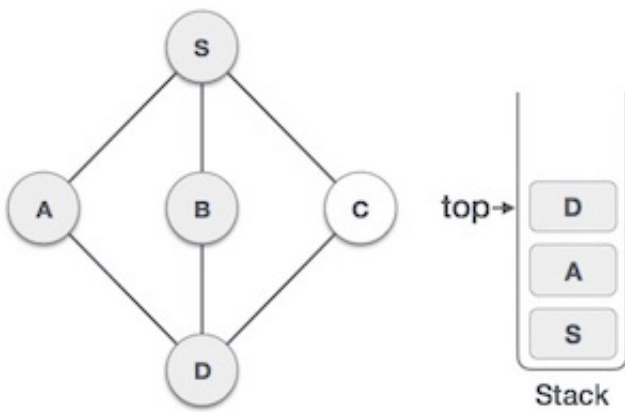
5.



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.

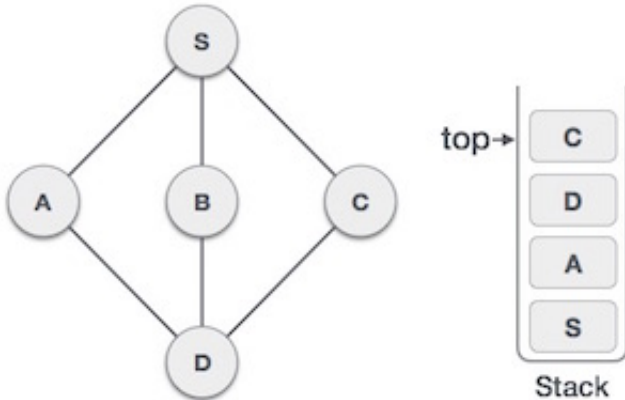
We check the stack top for return to the

6.



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.

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