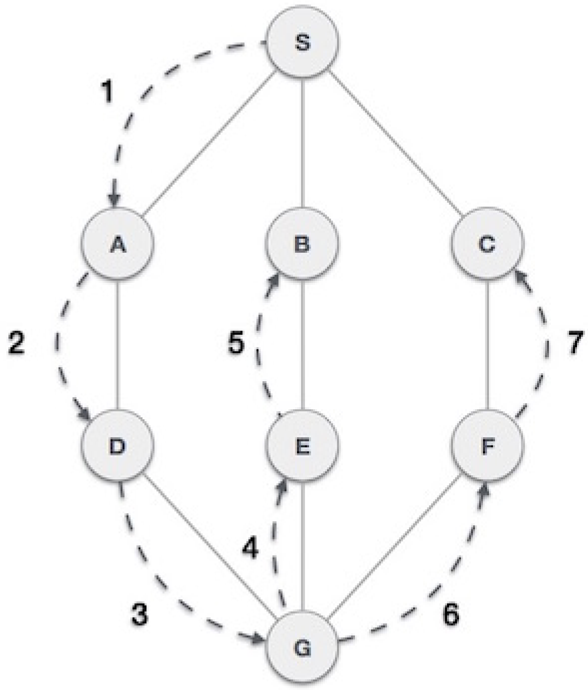
Depth First Search (DFS) Algorithm

Depth First Search (DFS) Algorithm

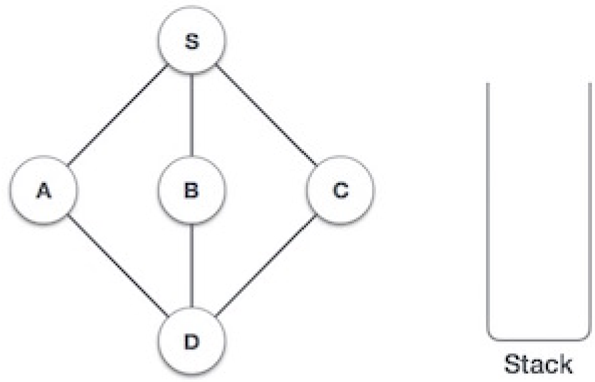
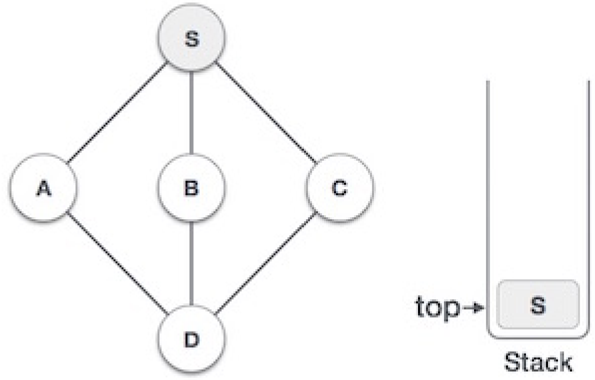
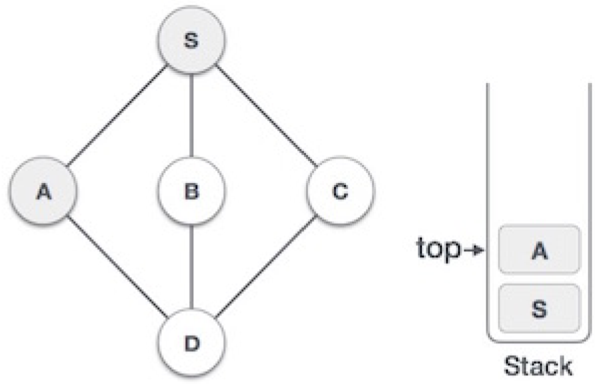
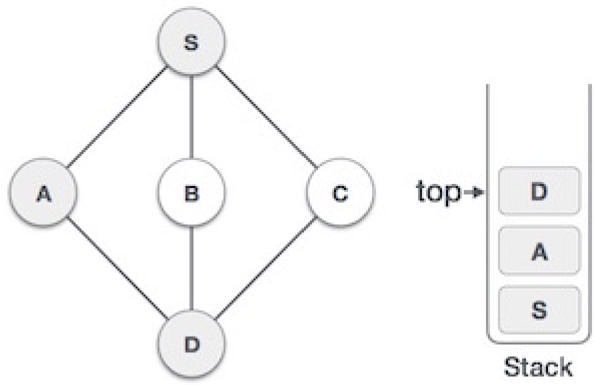
Depth First Search (DFS) algorithm is a recursive algorithm for searching all the vertices of a graph or tree data structure. This algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

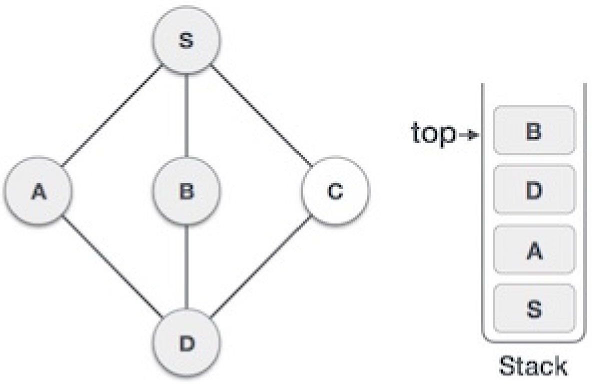
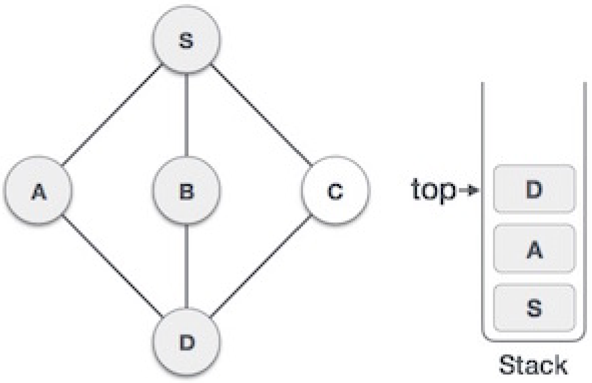
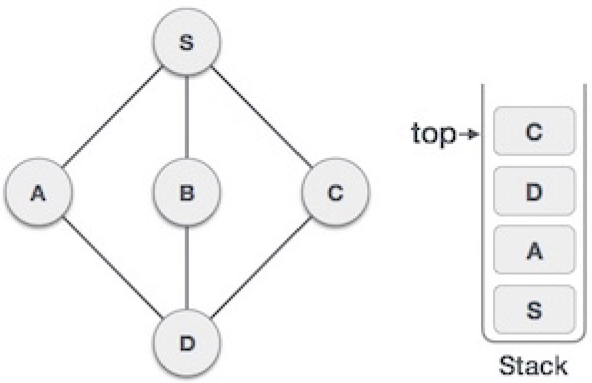


As in the example given above, DFS algorithm traverses from S to A to D to G to E to B first, then to F and lastly to C. It employs the following rules.

- **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 **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.
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.

Example

Following are the implementations of Depth First Search (DFS) Algorithm in various programming languages −

C | C++ | Java | Python

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 5
```

```
struct Vertex {
   char label;
   bool visited;
};
//stack variables
int stack[MAX];
int top = -1;
//graph variables
//array of vertices
struct Vertex* lstVertices[MAX];
//adjacency matrix
int adjMatrix[MAX][MAX];
//vertex count
int vertexCount = 0;
//stack functions
void push(int item) {
   stack[++top] = item;
}
int pop() {
   return stack[top--];
}
int peek() {
   return stack[top];
}
bool isStackEmpty() {
   return top == -1;
}
//graph functions

//add vertex to the vertex list
void addVertex(char label) {
   struct Vertex* vertex = (struct Vertex*) malloc(sizeof(struct Vertex))
   vertex->label = label;
   vertex->visited = false;
   lstVertices[vertexCount++] = vertex;
}
//add edge to edge array
void addEdge(int start,int end) {
   adjMatrix[start][end] = 1;
   adjMatrix[end][start] = 1;
}
//display the vertex
```

```c
void displayVertex(int vertexIndex) {
   printf("%c ",lstVertices[vertexIndex]->label);
}
//get the adjacent unvisited vertex
int getAdjUnvisitedVertex(int vertexIndex) {
   int i;
   for(i = 0; i < vertexCount; i++) {
      if(adjMatrix[vertexIndex][i] == 1 && lstVertices[i]->visited == fal
         return i;
      }
   }
   return -1;
}
void depthFirstSearch() {
   int i;
   //mark first node as visited
   lstVertices[0]->visited = true;
   //display the vertex
   displayVertex(0);
   //push vertex index in stack
   push(0);
   while(!isStackEmpty()) {
      //get the unvisited vertex of vertex which is at top of the stack
      int unvisitedVertex = getAdjUnvisitedVertex(peek());
      //no adjacent vertex found
      if(unvisitedVertex == -1) {
         pop();
      } else {
         lstVertices[unvisitedVertex]->visited = true;
         displayVertex(unvisitedVertex);
         push(unvisitedVertex);
      }
   }
   //stack is empty, search is complete, reset the visited flag
   for(i = 0;i < vertexCount;i++) {
      lstVertices[i]->visited = false;
   }
}
int main() {
   int i, j;

   for(i = 0; i < MAX; i++) {   // set adjacency
```

```
        for(j = 0; j < MAX; j++) // matrix to 0
            adjMatrix[i][j] = 0;
    }
    addVertex('S');   // 0
    addVertex('A');   // 1
    addVertex('B');   // 2
    addVertex('C');   // 3
    addVertex('D');   // 4
    addEdge(0, 1);    // S - A
    addEdge(0, 2);    // S - B
    addEdge(0, 3);    // S - C
    addEdge(1, 4);    // A - D
    addEdge(2, 4);    // B - D
    addEdge(3, 4);    // C - D
    printf("Depth First Search: ");
    depthFirstSearch();
    return 0;
}
```

Output

```
Depth First Search: S A D B C
```

Click to check C implementation of Depth First Search (BFS) Algorithm

Complexity of DFS Algorithm

Time Complexity

The time complexity of the DFS algorithm is represented in the form of O(V + E), where V is the number of nodes and E is the number of edges.

Space Complexity

The space complexity of the DFS algorithm is O(V).