Graph Traversal

Graph traversal is a technique used for searching a vertex in a graph. The graph traversal is also used to decide the order of vertices is 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 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...

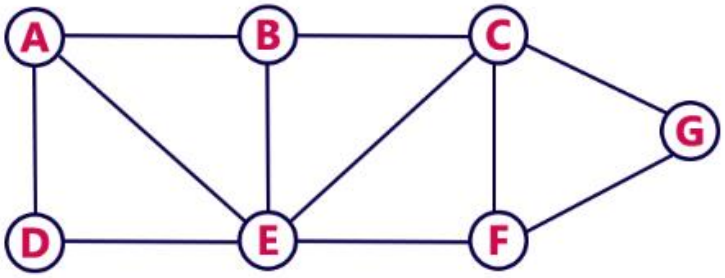
- **Step 1 -** Define a Stack of size total number of vertices in the graph.
- **Step 2 -** Select any vertex as **starting point** for traversal. Visit that vertex and push it on to the Stack.
- **Step 3 -** Visit any one of the non-visited **adjacent** vertices of a vertex which is at the top of stack and push it on to 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 **back tracking** 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

Back tracking is coming back to the vertex from which we reached the current vertex.

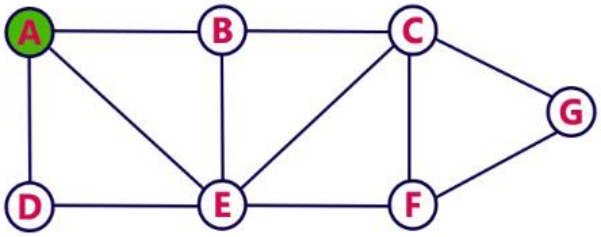
Example:

Consider the following example graph to perform DFS traversal



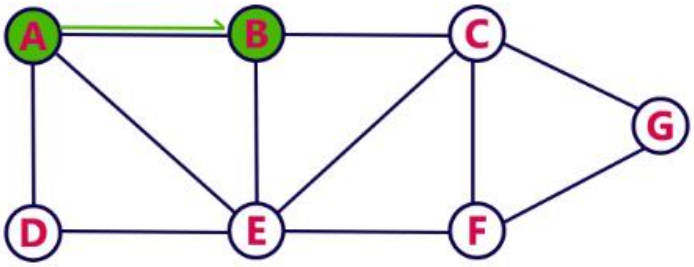
Step 1:

- Select the vertex **A** as starting point (visit **A**).
- Push **A** on to the Stack.



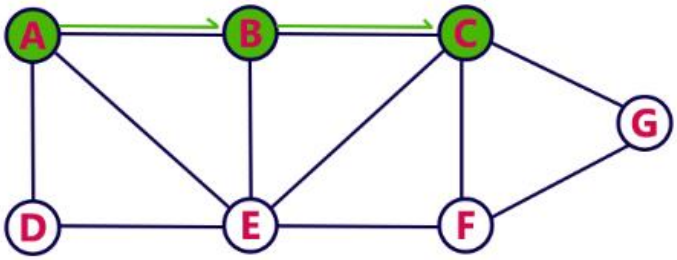
Step 2:

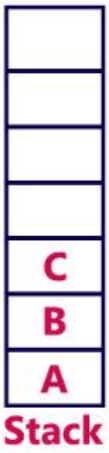
- Visit any adjacent vertex of **A** which is not visited (**B**).
- Push newly visited vertex B on to the Stack.



Step 3:

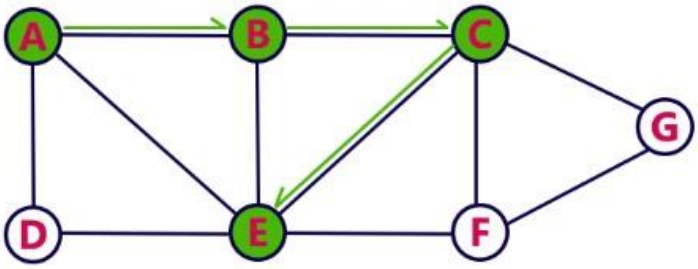
- Visit any adjacent vertext of **B** which is not visited (**C**).
- Push C on to the Stack.

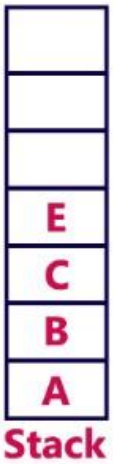




Step 4:

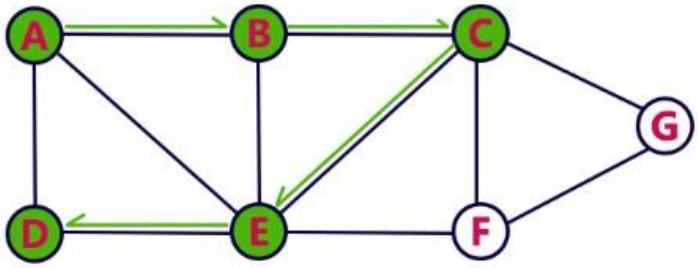
- Visit any adjacent vertext of **C** which is not visited (**E**).
- Push E on to the Stack





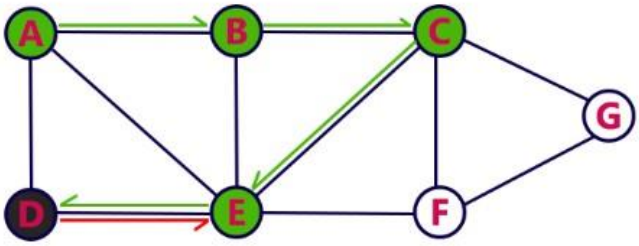
Step 5:

- Visit any adjacent vertext of **E** which is not visited (**D**).
- Push D on to the Stack



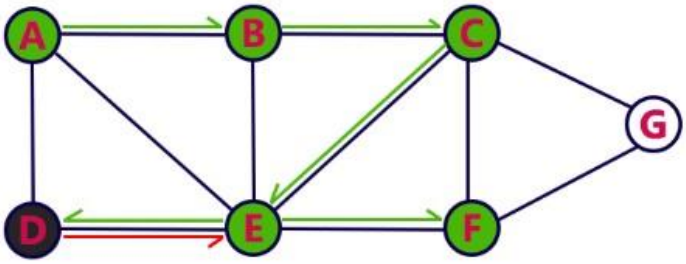
Step 6:

- There is no new vertiex to be visited from D. So use back track.
- Pop D from the Stack.



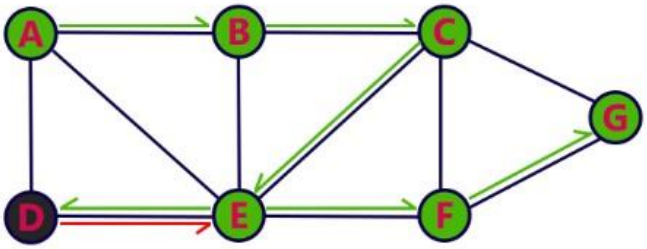
Step 7:

- Visit any adjacent vertex of **E** which is not visited (**F**).
- Push **F** on to the Stack.



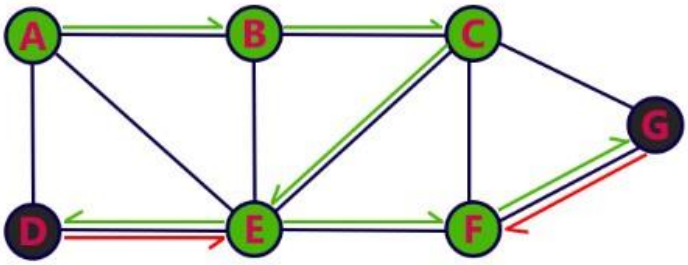
Step 8:

- Visit any adjacent vertex of **F** which is not visited (**G**).
- Push **G** on to the Stack.



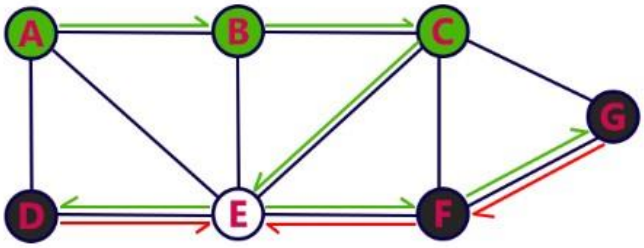
Step 9:

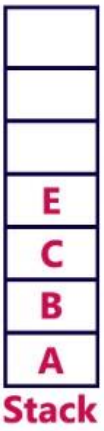
- There is no new vertiex to be visited from G. So use back track.
- Pop G from the Stack.



Step 10:

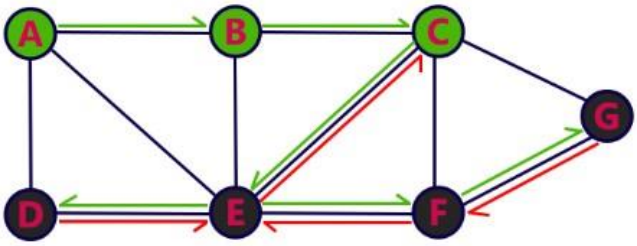
- There is no new vertiex to be visited from F. So use back track.
- Pop F from the Stack.





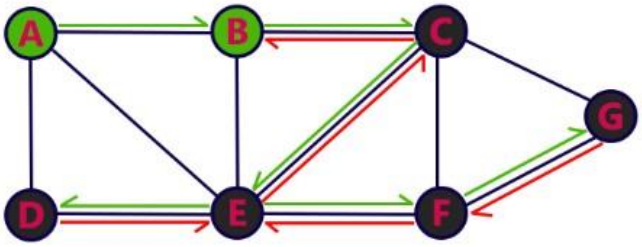
Step 11:

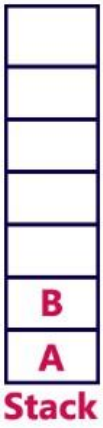
- There is no new vertiex to be visited from E. So use back track.
- Pop E from the Stack.



Step 12:

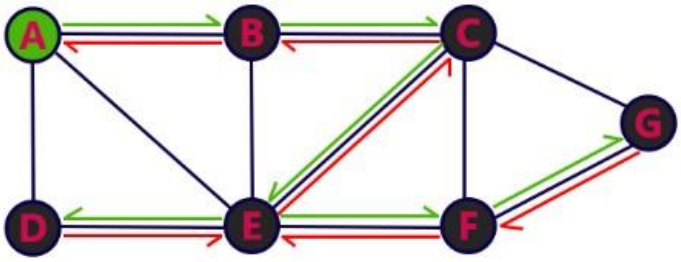
- There is no new vertiex to be visited from C. So use back track.
- Pop C from the Stack.





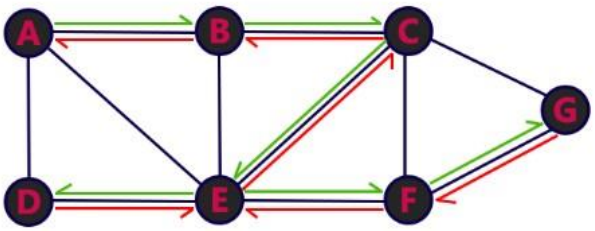
Step 13:

- There is no new vertiex to be visited from B. So use back track.
- Pop B from the Stack.

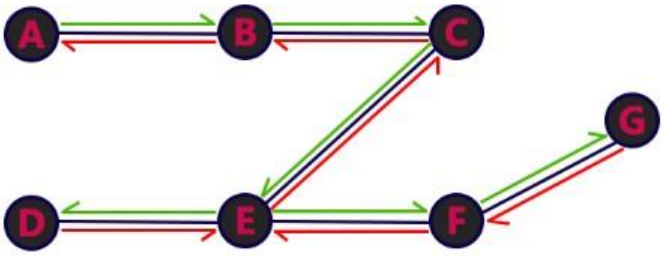


Step 14:

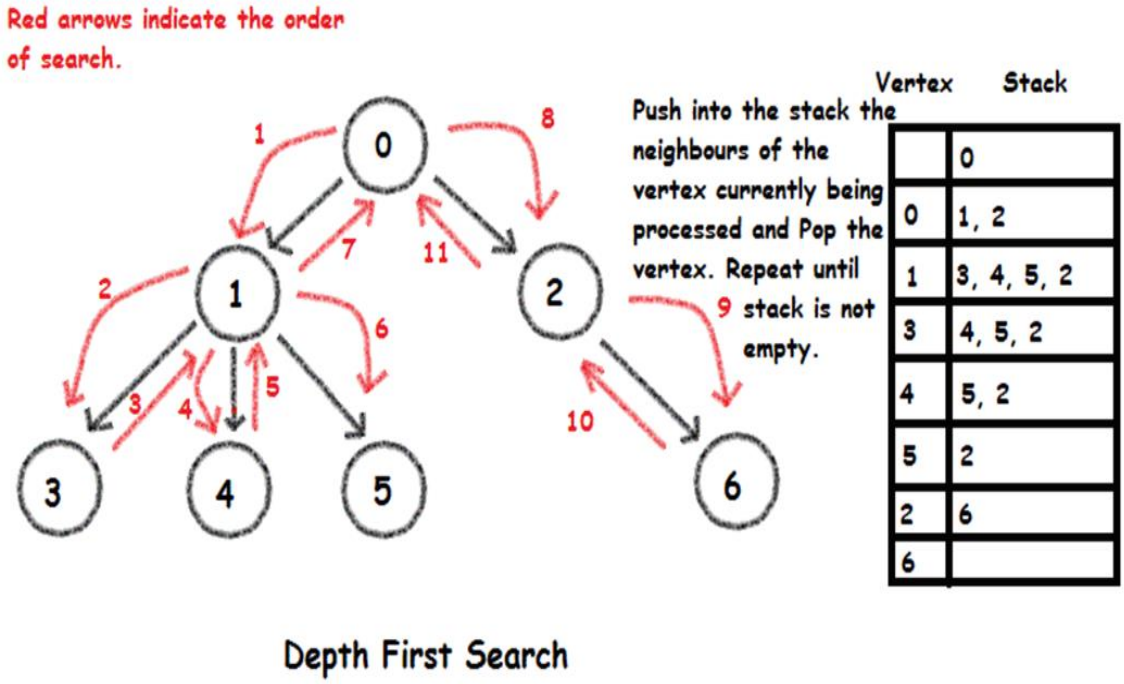
- There is no new vertiex to be visited from A. So use back track.
- Pop A from the Stack.



- Stack became Empty. So stop DFS Treversal.
- Final result of DFS traversal is following spanning tree.



Example 2:



Depth First Search

Applications of DFS:

Following are the problems that use DFS as a building block.

1) For an unweighted graph, DFS traversal of the graph produces the minimum spanning tree and all pair shortest path tree. DFS is at the heart of Prims and Kruskals algorithms.

2) **Detecting cycle in a graph**
A graph has cycle if and only if we see a back edge during DFS. So we can run DFS for the graph and check for back edges.

3) **Path Finding**

DFS algorithm can be used to find a path between two given vertices u and z.
i) Call DFS(G, u) with u as the start vertex.
ii) Use a stack S to keep track of the path between the start vertex and the current vertex.
iii) As soon as destination vertex z is encountered, return the path as the
contents of the stack

4) **Topological Sorting**
DFS is an intermediate step for topological sorting.

5) Finding Strongly Connected Components of a graph A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex. (See this for DFS based algo for finding Strongly Connected Components)

6) DFS is very helpful in solving almost all the maze puzzles. Most of the maze puzzles can be converted into Graph problems and traversals result into the solution. DFS can be adapted to find all solutions to a maze by only including nodes on the current path in the visited set.