Advanced Algorithms & Data Structures

Department of CSE

ADVANCED ALGORITHMS AND DATA STRUCTURES
23CS03HF

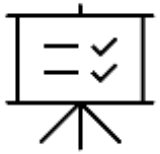
Topic:

GRAPH TRAVERSALS ALGORITHMS: BFS AND DFS

AIM OF THE SESSION

To familiarize students with the concept of graph traversal techniques.

INSTRUCTIONAL OBJECTIVES



This Session is designed to:

1. Demonstrate :- graph traversal techniques.
2. Describe :- DFS and BFS.

LEARNING OUTCOMES

At the end of this session, you should be able to:

1. Define :- graph traversal techniques.
2. Describe :- DFS and BFS.
3. Summarize:- graph traversal techniques.

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)

Breadth First Search

- **BFS** Stands for Breadth First Search. It is also known as level order traversal.
- The Queue data structure is used for the Breadth First Search traversal.
- When we use the BFS algorithm for the traversal in a graph, we can consider any node as a root node.
- **BFS traversal of a graph produces a spanning tree as final result. Spanning Tree is a graph without loops.**

Breadth First Search Algorithm

Step 1 - Define a Queue of size total number of vertices in the graph.

Step 2 - Select any vertex as starting point for traversal. Visit that vertex and insert it into the Queue.

Step 3 - Visit all the non-visited adjacent vertices of the vertex which is at front of the Queue and insert them into the Queue.

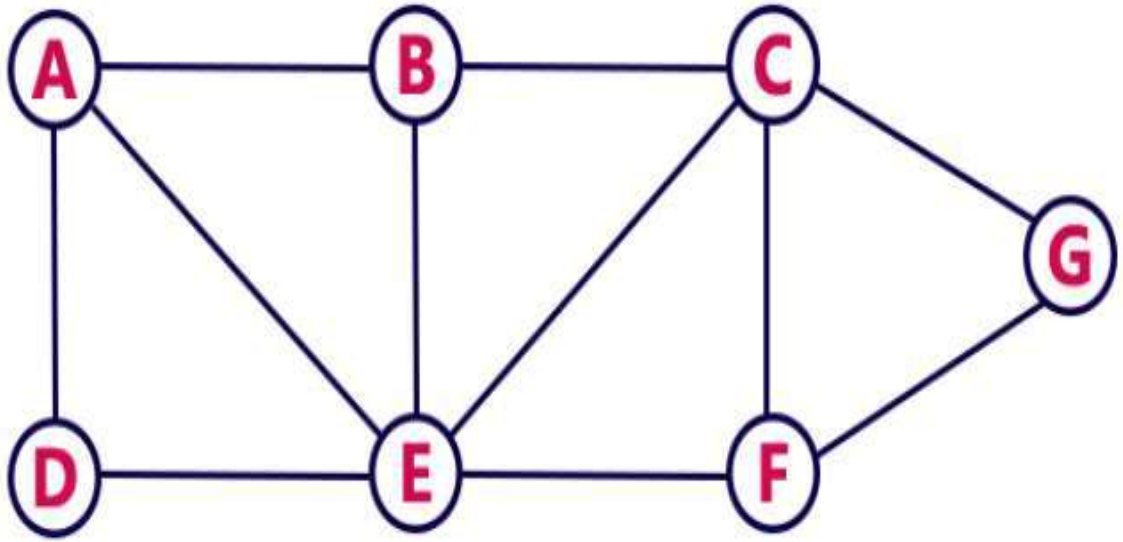
Step 4 - When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.

Step 5 - Repeat steps 3 and 4 until queue becomes empty.

Step 6 - When queue becomes empty, then produce final spanning tree by removing unused edges from the graph

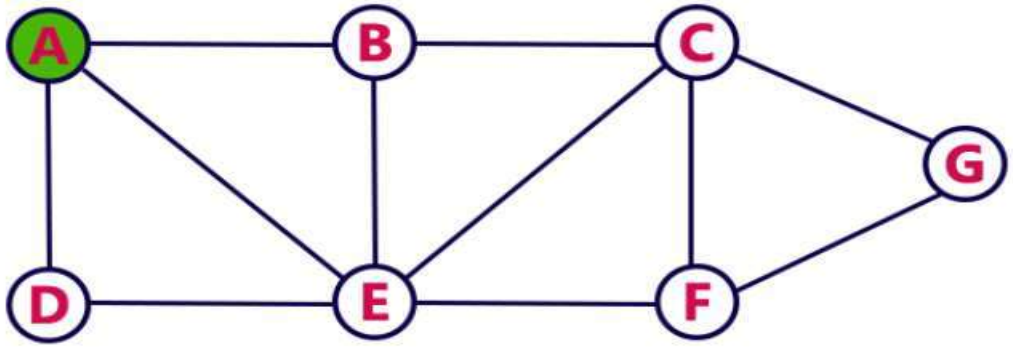
Example

Consider the following example graph to perform BFS traversal



Step 1:

- Select the vertex **A** as starting point (visit **A**).
- Insert **A** into the Queue.

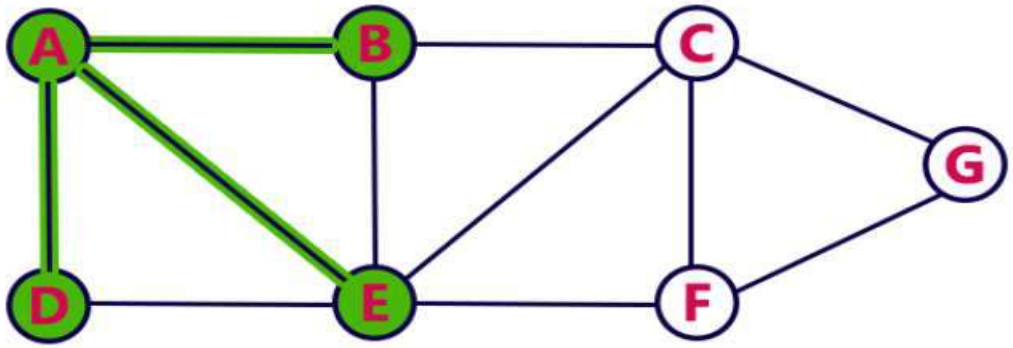


Queue

A						

Step 2:

- Visit all adjacent vertices of **A** which are not visited (**D**, **E**, **B**).
- Insert newly visited vertices into the Queue and delete A from the Queue..

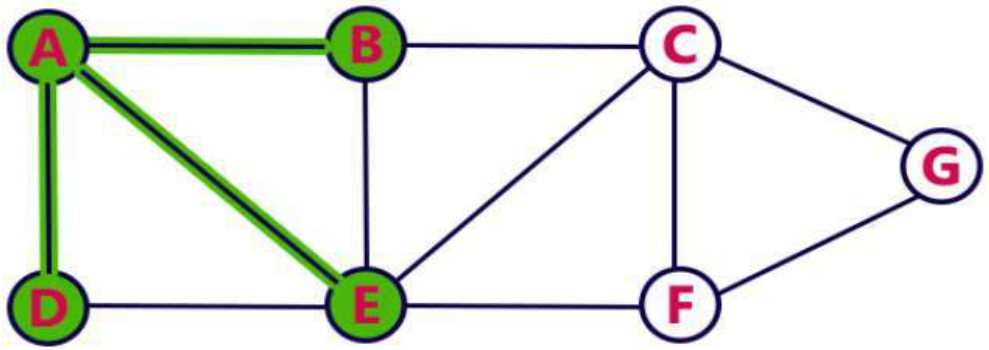


Queue

	D	E	B			

Step 3:

- Visit all adjacent vertices of **D** which are not visited (there is no vertex).
- Delete D from the Queue.

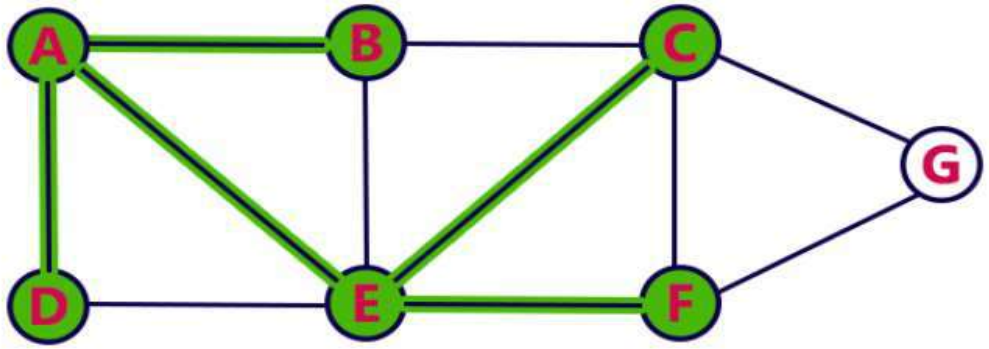


Queue

		E	B			

Step 4:

- Visit all adjacent vertices of **E** which are not visited (**C**, **F**).
- Insert newly visited vertices into the Queue and delete E from the Queue.

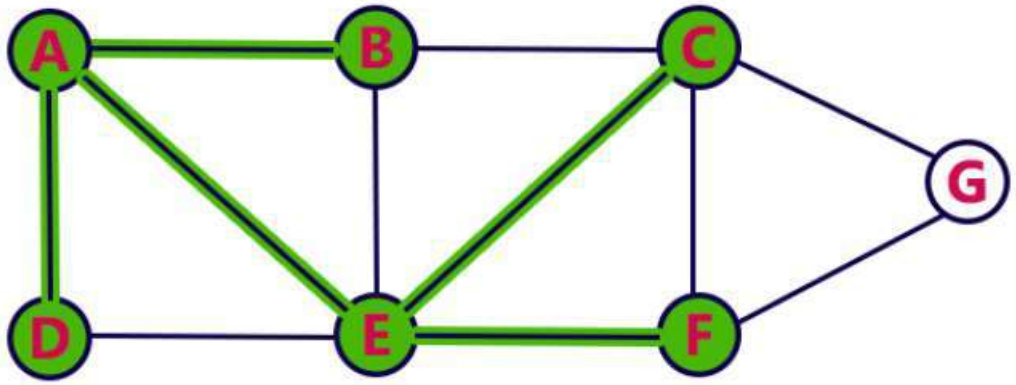


Queue

			B	C	F	

Step 5:

- Visit all adjacent vertices of **B** which are not visited (**there is no vertex**).
- Delete **B** from the Queue.

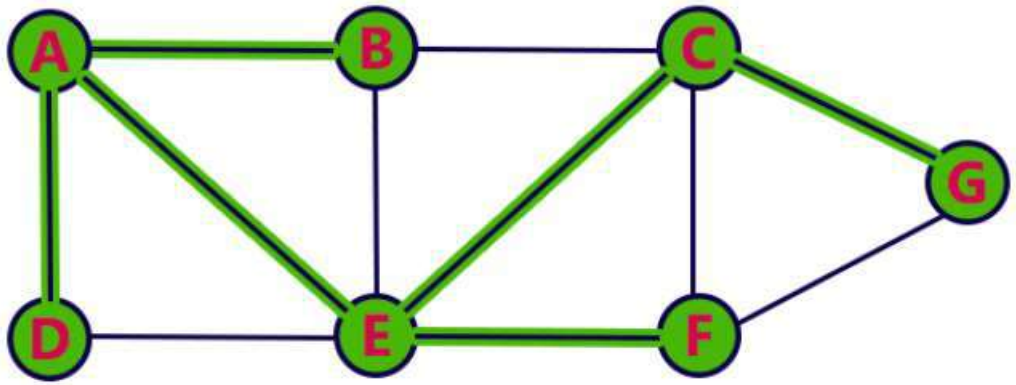


Queue

				C	F	

Step 6:

- Visit all adjacent vertices of **C** which are not visited (**G**).
- Insert newly visited vertex into the Queue and delete **C** from the Queue.

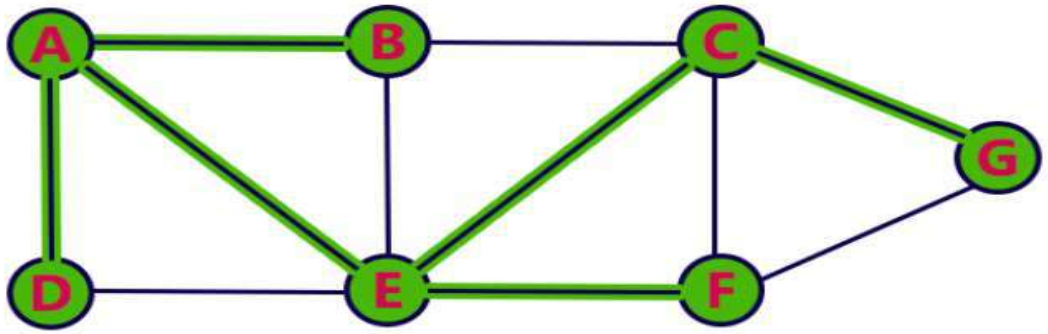


Queue

					F	G

Step 7:

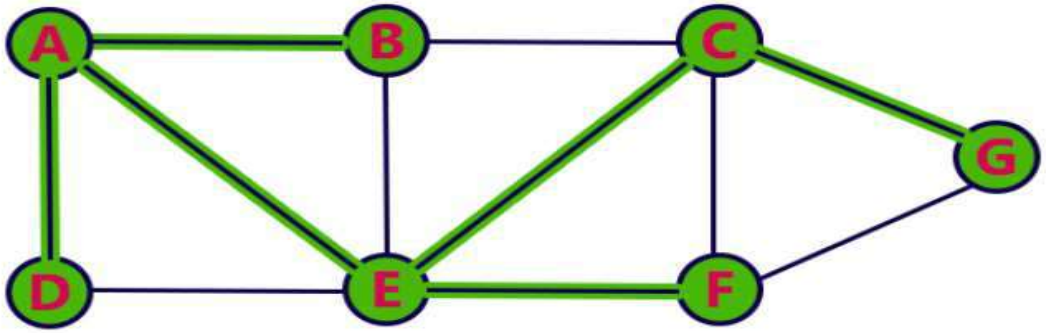
- Visit all adjacent vertices of **F** which are not visited (**there is no vertex**).
- Delete **F** from the Queue.



Queue

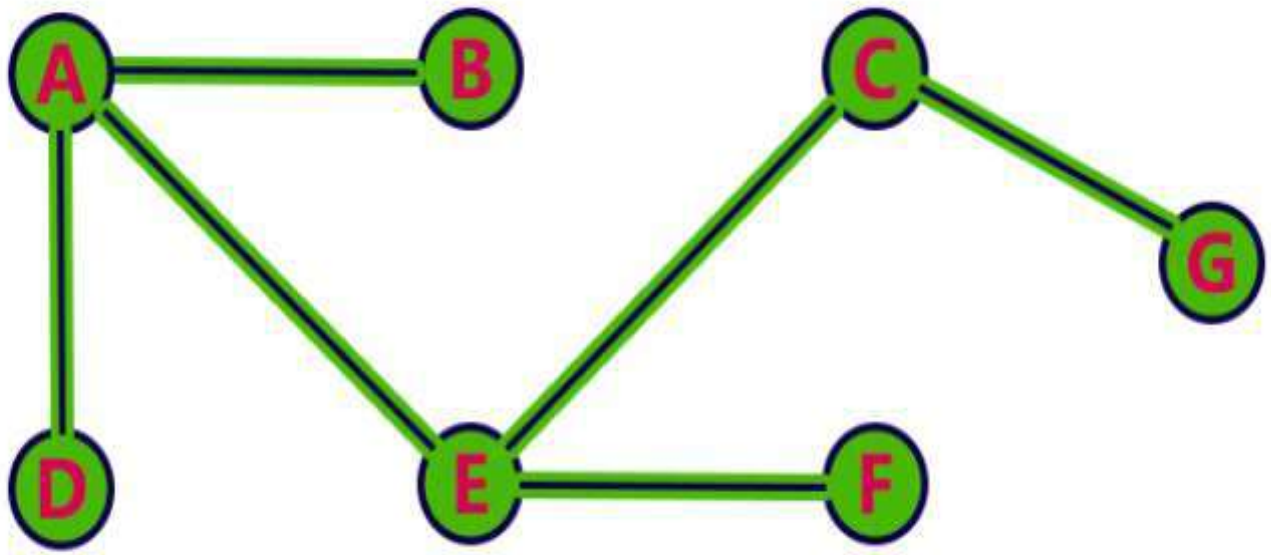
Step 8:

- Visit all adjacent vertices of **G** which are not visited (**there is no vertex**).
- Delete **G** from the Queue.



Queue

- Queue became Empty. So, stop the BFS process.
- Final result of BFS is a Spanning Tree as shown below...



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.
- It is also called as Preorder Traversal.

Depth First Search Algorithm

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.

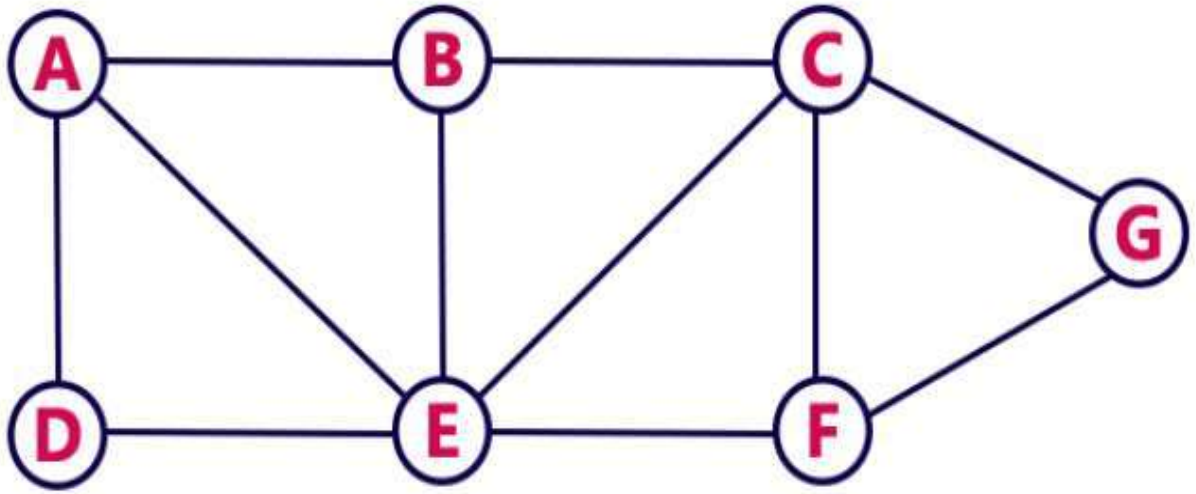
Step 5 - When there is no new vertex to visit then use back tracking and pop one vertex from the stack.

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

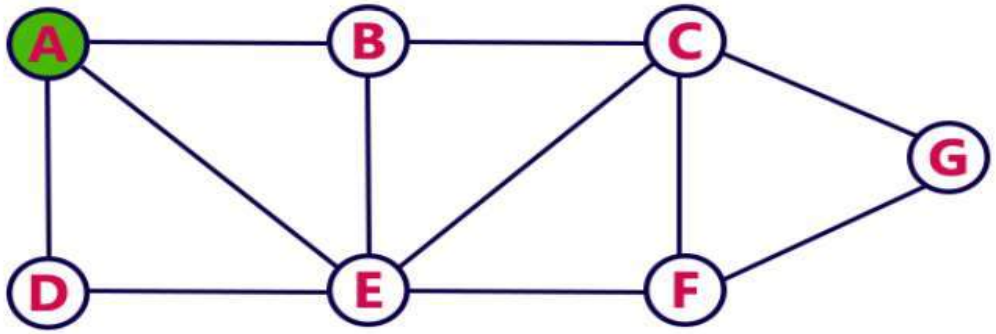
Example

Consider the following example graph to perform DFS traversal



Step 1:

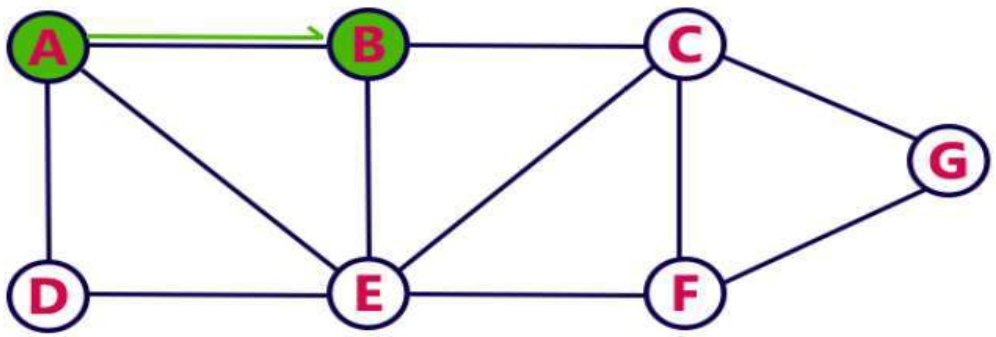
- Select the vertex **A** as starting point (visit **A**).
- Push **A** on to the Stack.



Stack

Step 2:

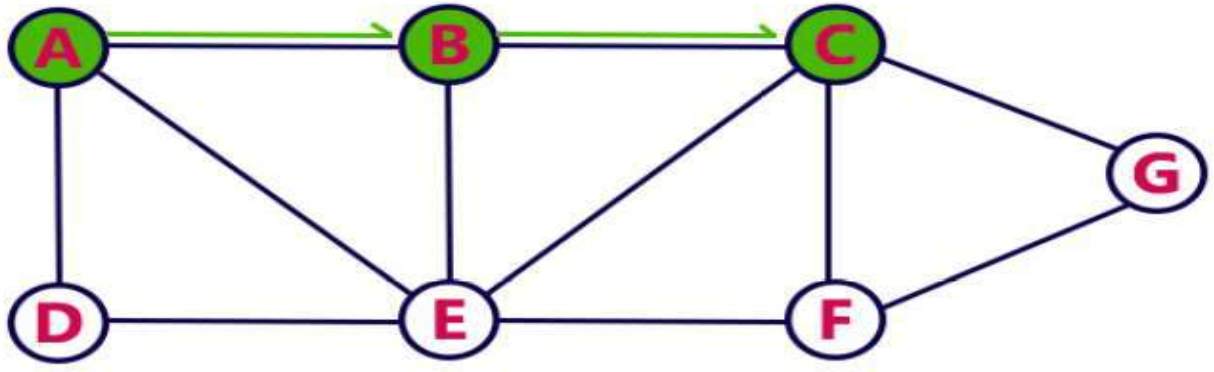
- Visit any adjacent vertex of **A** which is not visited (**B**).
- Push newly visited vertex B on to the Stack.

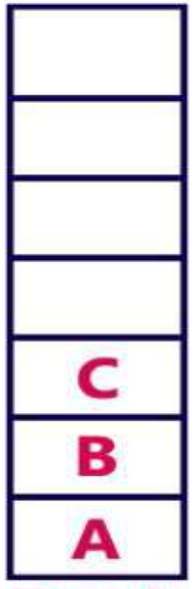


Stack

Step 3:

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

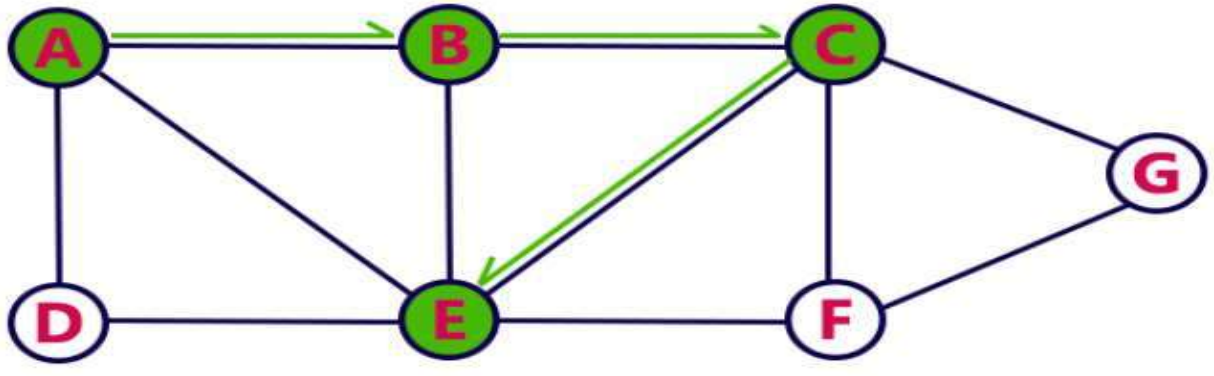


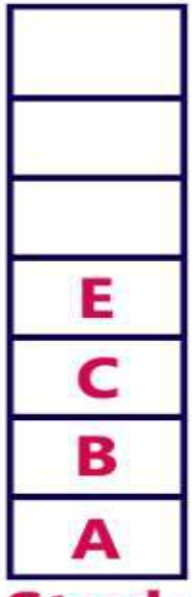


Stack

Step 4:

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

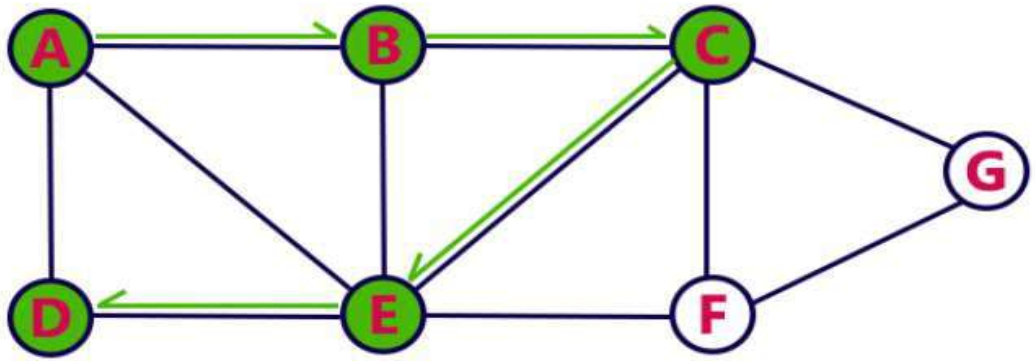


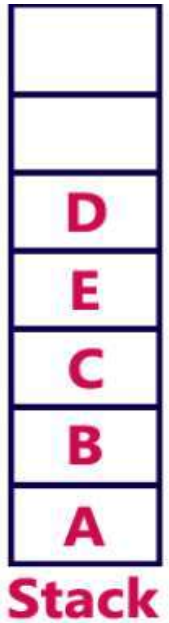


Stack

Step 5:

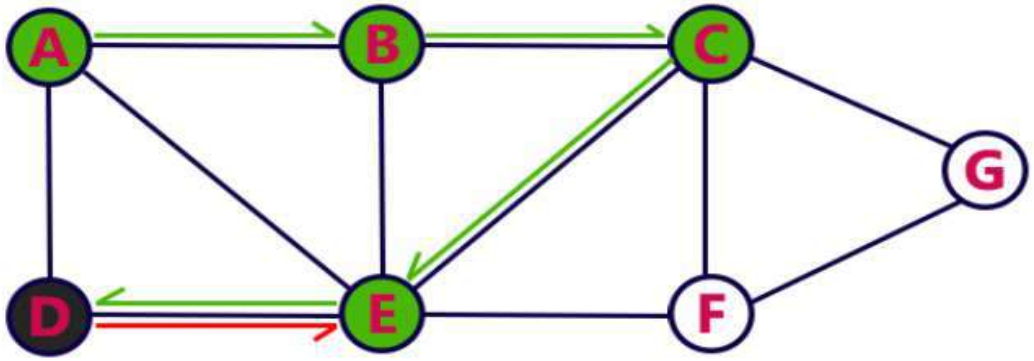
- Visit any adjacent vertext of **E** which is not visited (**D**).
- Push D on to the Stack

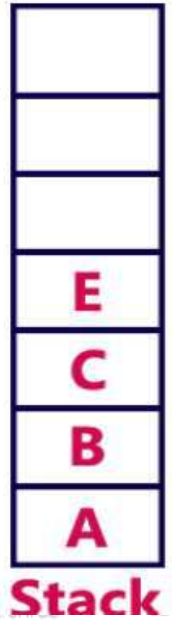




Step 6:

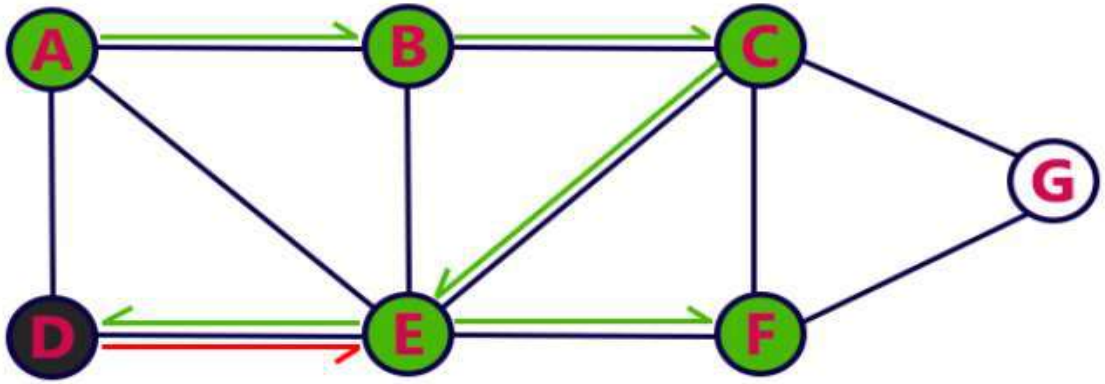
- There is no new vertiex to be visited from D. So use back track.
- Pop D from the Stack.





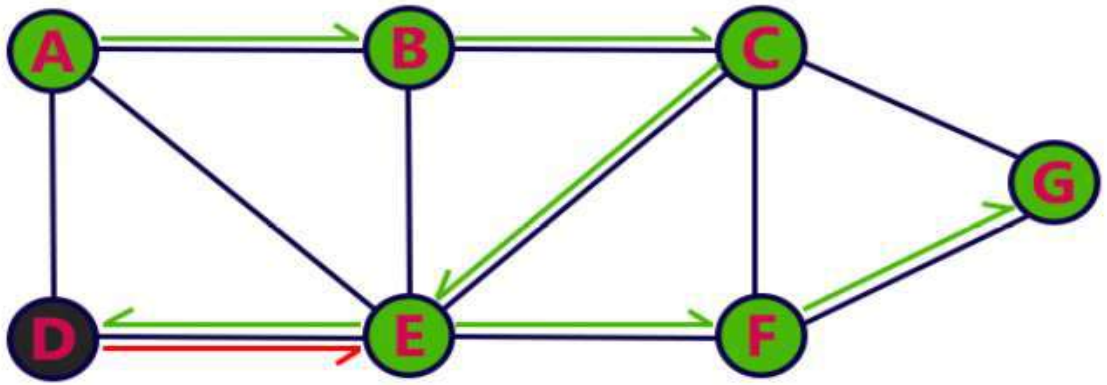
Step 7:

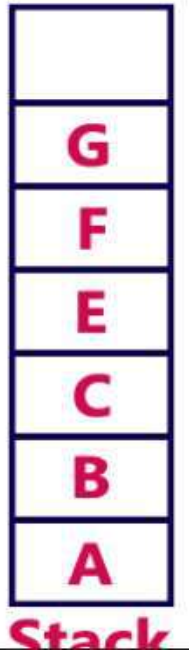
- Visit any adjacent vertex of **E** which is not visited (**F**).
- Push **F** on to the Stack.



Step 8:

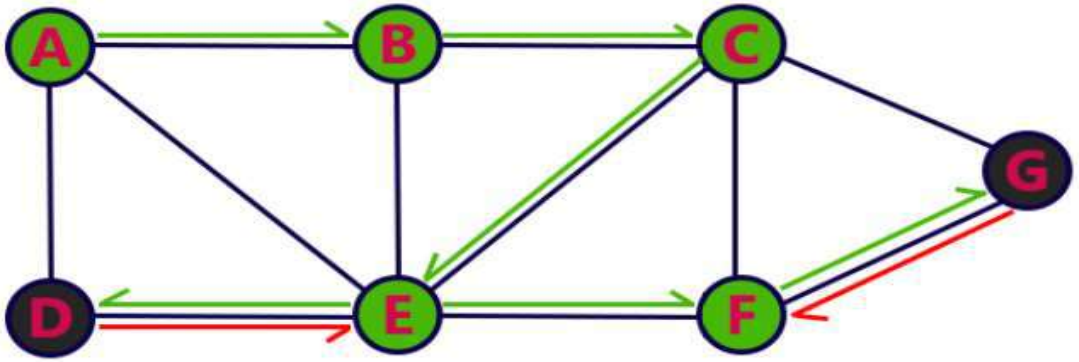
- Visit any adjacent vertex of **F** which is not visited (**G**).
- Push **G** on to the Stack.

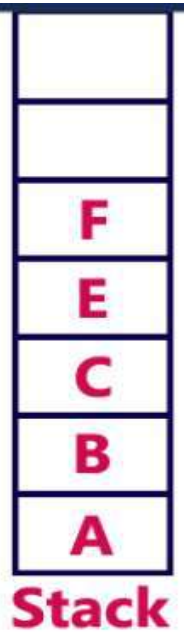




Step 9:

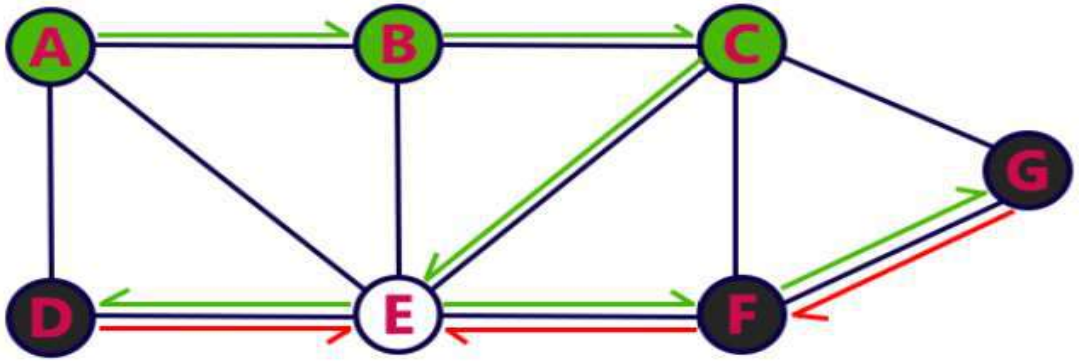
- There is no new vertiex to be visited from G. So use back track.
- Pop G from the Stack.

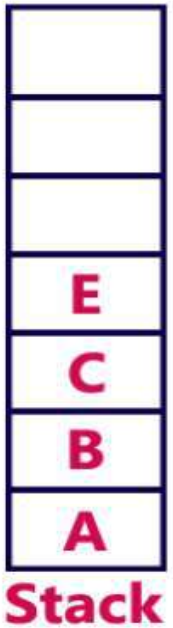




Step 10:

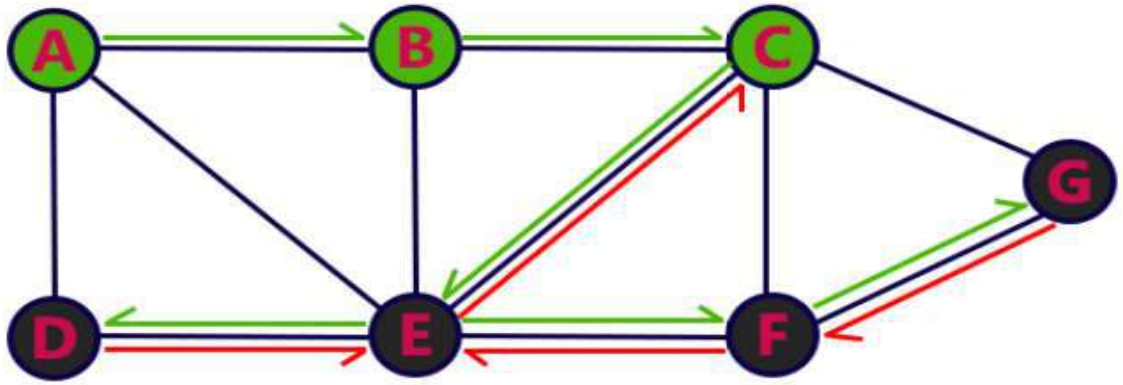
- There is no new vertiex to be visited from F. So use back track.
- Pop F from the Stack.

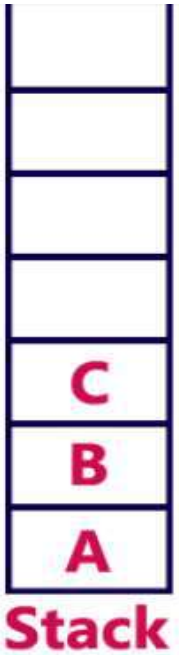




Step 11:

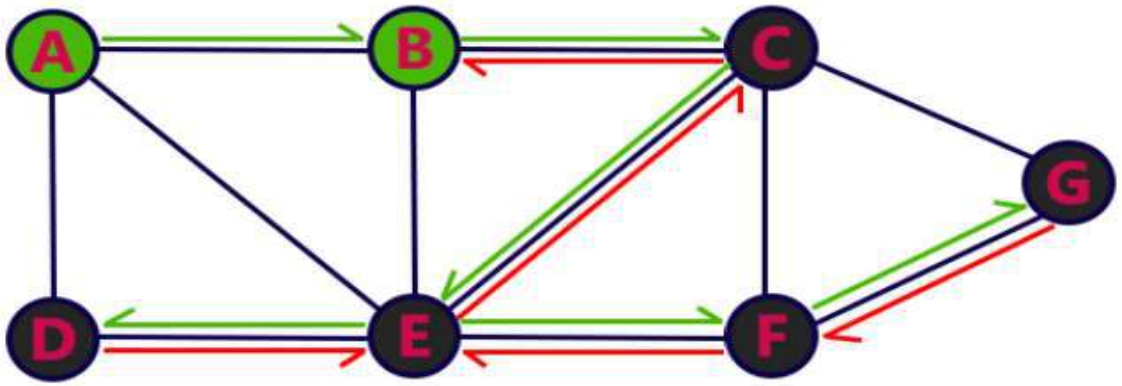
- There is no new vertiex to be visited from E. So use back track.
- Pop E from the Stack.

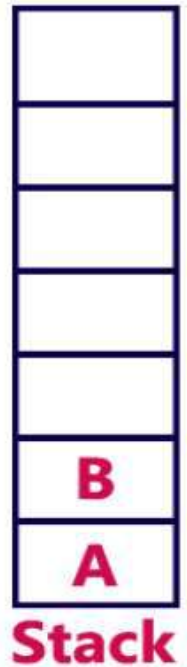




Step 12:

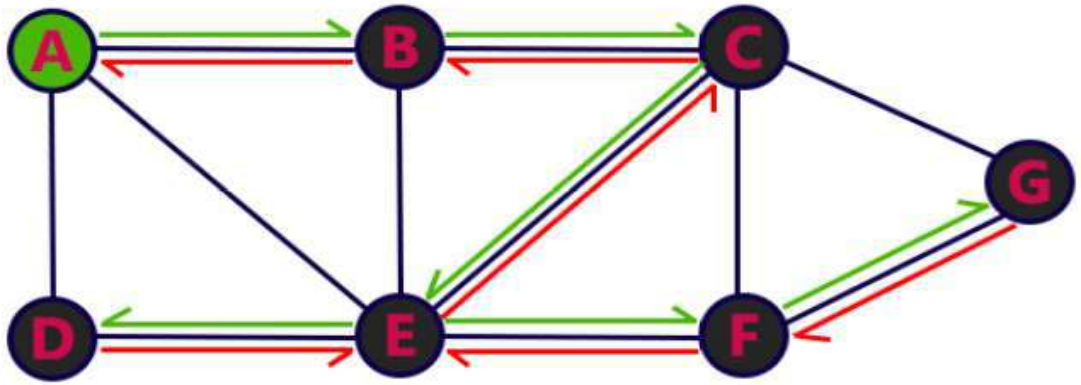
- There is no new vertiex to be visited from C. So use back track.
- Pop C from the Stack.

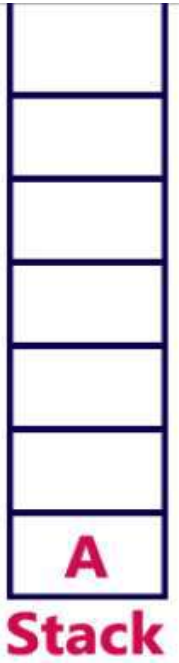




Step 13:

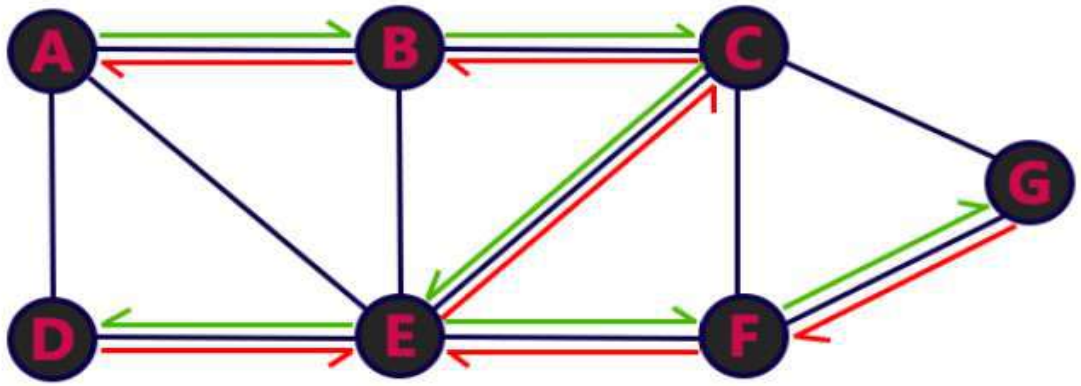
- There is no new vertiex to be visited from B. So use back track.
- Pop B from the Stack.





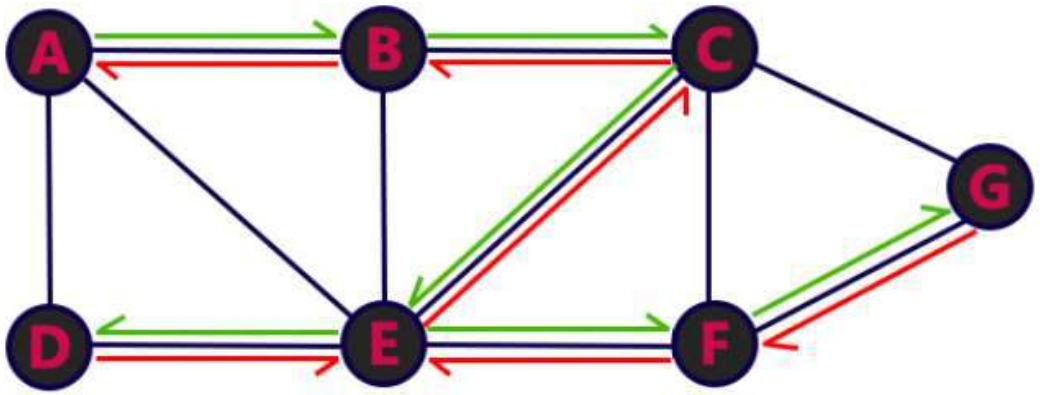
Step 14:

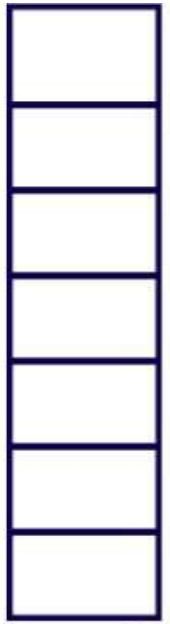
- There is no new vertiex to be visited from A. So use back track.
- Pop A from the Stack.



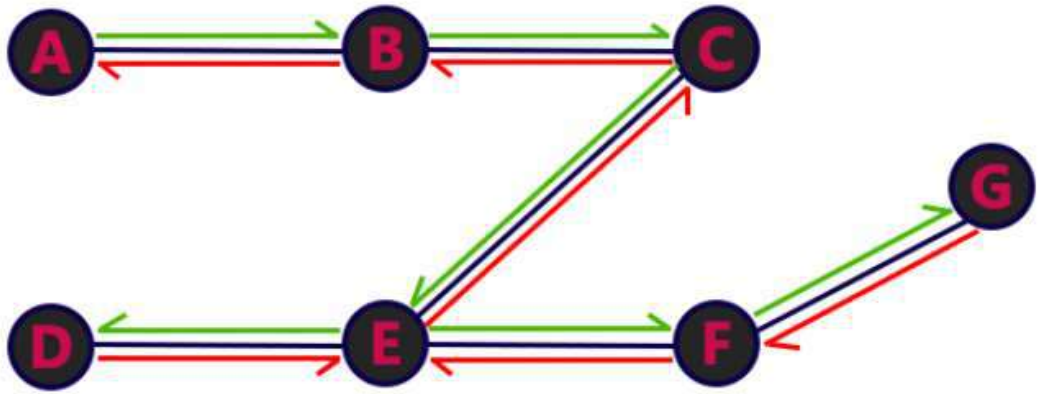
Step 14:

- 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.



Time Complexity of BFS and DFS

- Time Complexity of BFS = O(V+E) where V is vertices and E is edges.
- Time Complexity of DFS is also O(V+E) where V is vertices and E is edges.

SUMMARY

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.

SELF-ASSESSMENT QUESTIONS

- What is the time complexity of the Ford-Fulkerson algorithm in the worst case if implemented with the breadth-first search (BFS)?

A. (O(V + E))

B. (O(V^2E))

C. (O(VE^2))

D. (O(V^3))

- What does the Ford-Fulkerson algorithm use to find the augmenting path?

A. Depth-First Search (DFS)

B. Breadth-First Search (BFS)

C. Either DFS or BFS

D. Dijkstra's Algorithm

TERMINAL QUESTIONS

- **What are BFS and DFS with examples?**
- **How to choose between BFS and DFS?**
- **Which is faster, BFS or DFS?**

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books :

1Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein., 3rd, 2009, The MIT Press.

2 Algorithm Design Manual, Steven S. Skiena., 2nd, 2008, Springer.

3 Data Structures and Algorithms in Python, Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser., 2nd, 2013, Wiley.

4 The Art of Computer Programming, Donald E. Knuth, 3rd, 1997, Addison-Wesley Professiona.

MOOCS :

1. https://www.coursera.org/specializations/algorithms?=

2.https://www.coursera.org/learn/dynamic-programming-greedy-algorithms#modules

THANK YOU