1- Depth First Search (DFS)

A depth first search of a connected graph G = (V, E) is a recursive algorithm to examine the vertices and the edges of a graph. To begin a depth first search, choose an adjacency list representation for **G** graph. Different adjacency list structures will give rise to examining the vertices and edges in different orders, but in any representation, each edge and vertex will be visited.

Algorithm Steps: -

Step 1: SET STATUS = 1 (ready state) for each node in **G.**

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

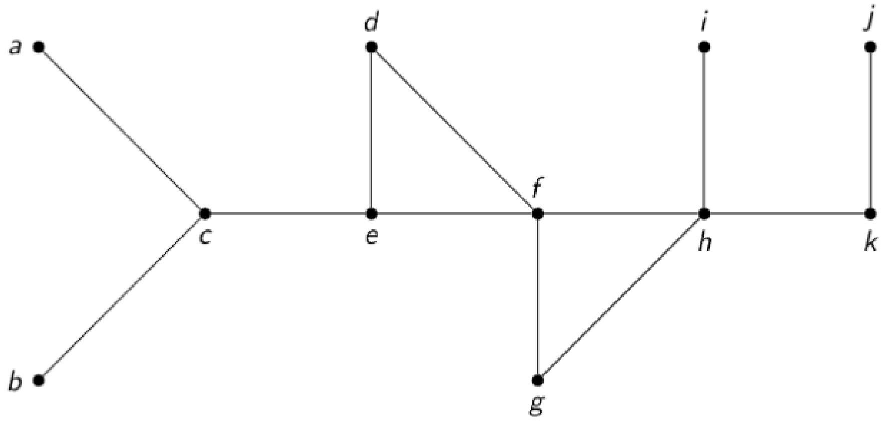
Step 3: Repeat Steps 4 and 5 until **STACK** is empty.

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state).

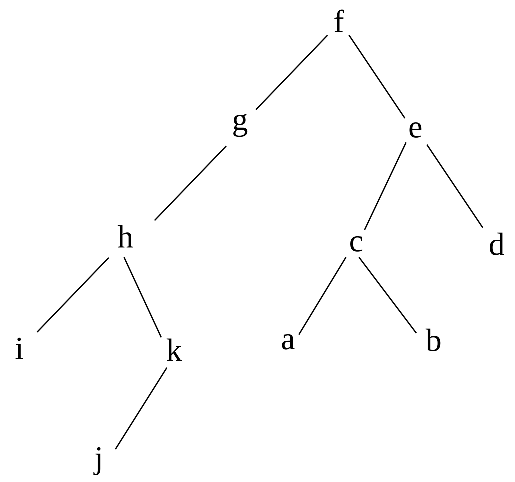
Step 5: Push on the stack all the neighbors of **N** that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state).

[END OF LOOP]

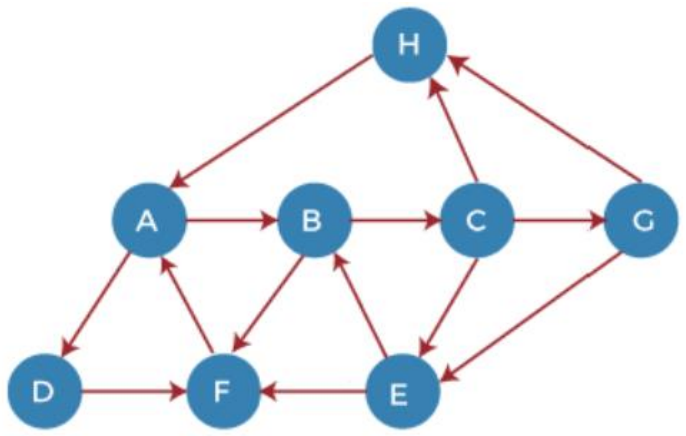
Example 1. Use **DFS** algorithm to find a **spanning tree** starting with (**f**) for the graph G shown below: -



Solution.



Example 2. In the example given below, there is a directed graph having 8 vertices. Start examining the graph starting from **Node H**.



Solution.

Step 1 - First, push H onto the stack

1. **STACK: H**

Step 2 - **POP** the top element from the stack, i.e., **H**, and print it. Now, **PUSH** all the neighbors of H onto the stack that are in ready state.

1. **Print: H**
2. **STACK: A**

Step 3 - POP the top element from the stack, i.e., A, and print it. Now, **PUSH** all the neighbors of **A** onto the stack that are in ready state.

1. **Print: A**
2. **STACK: B, D**

Step 4 - POP the top element from the stack, i.e., D, and print it. Now, **PUSH** all the neighbors of D onto the stack that are in ready state.

1. **Print: D**
2. **STACK: B, F**

Step 5 - POP the top element from the stack, i.e., F, and print it. Now, **PUSH** all the neighbors of **F** onto the stack that are in ready state.

1. **Print: F**
2. **STACK: B**

Step 6 - POP the top element from the stack, i.e., **B**, and print it. Now, **PUSH** all the neighbors of **B** onto the stack that are in ready state.

1. **Print: B**
2. **STACK: C**

Step 7 - POP the top element from the stack, i.e., C, and print it. Now, **PUSH** all the neighbors of **C** onto the stack that are in ready state.

1. **Print: C**
2. **STACK: E, G**

Step 8 - POP the top element from the stack, i.e., **G** and **PUSH** all the neighbors of **G** onto the stack that are in ready state.

1. **Print: G**
2. **STACK: E**

Step 9 - **POP** the top element from the stack, i.e., **E** and **PUSH** all the neighbors of **E** onto the stack that are in ready state.

1. **Print: E**
2. **STACK:**

			D	F			G		
	H	A	B	B	B	C	E	E	

2- Breath First Search (BFS)

Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

There are many ways to traverse the graph, but among them, BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.

The steps involved in the **BFS** algorithm to explore a graph are given as follows -

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until **QUEUE** is empty.

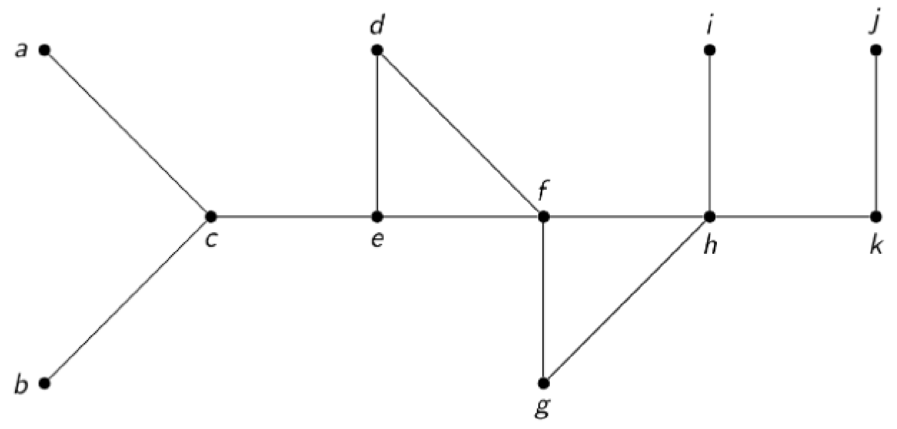
Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2

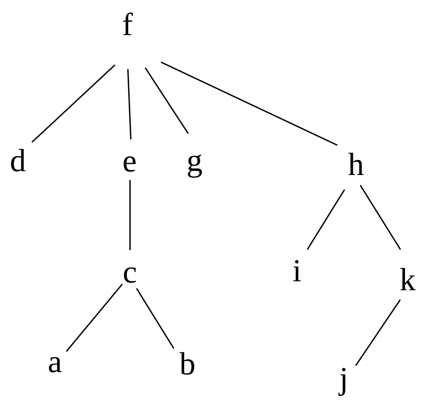
(waiting state)

[END OF LOOP]

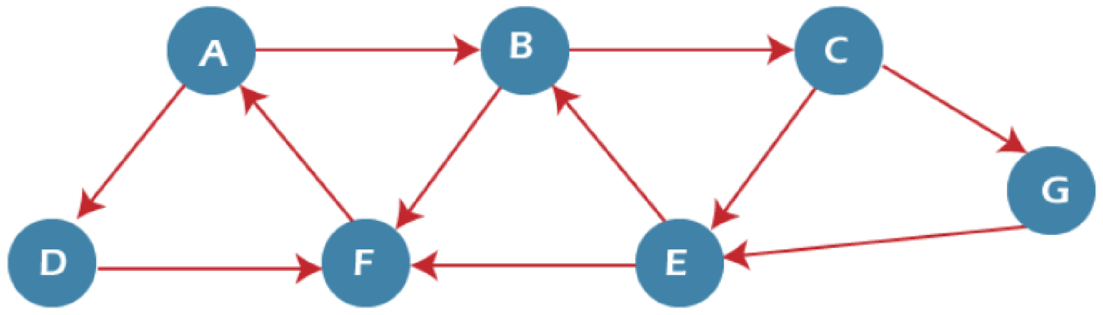
Example 3. Use **BFS** to find a **spanning tree** starting with (f) for the below graph.



Solution.



Example 4. In the example given below, there is a directed graph having 7 vertices. Start examining the graph starting from **Node A**.



Solution.

Step 1 - First, add A to queue1 and **NULL** to queue2.

1. **QUEUE1 = {A}**
2. **QUEUE2 = {NULL}**

Step 2 - Now, delete node A from queue1 and add it into queue2. Insert all neighbors of node A to queue1.

1. **QUEUE1 = {B, D}**
2. **QUEUE2 = {A}**

Step 3 - Now, delete node B from queue1 and add it into queue2. Insert all neighbors of node B to queue1.

1. **QUEUE1 = {D, C, F}**
2. **QUEUE2 = {A, B}**

Step 4 - Now, delete node D from queue1 and add it into queue2. Insert all neighbors of node D to queue1. The only neighbor of Node D is F since it is already inserted, so it will not be inserted again.

1. **QUEUE1 = {C, F}**
2. **QUEUE2 = {A, B, D}**

Step 5 - Delete node C from queue1 and add it into queue2. Insert all neighbors of node C to queue1.

1. **QUEUE1 = {F, E, G}**
2. **QUEUE2 = {A, B, D, C}**

Step 6 - Delete node **F** from queue1 and add it into queue2. Insert all neighbors of node F to queue1. Since all the neighbors of node F are already present, we will not insert them again.

1. **QUEUE1 = {E, G}**
2. **QUEUE2 = {A, B, D, C, F}**

Step 7 - Delete node E from queue1. Since all of its neighbors have already been added, so we will not insert them again. Now, all the nodes are visited, and the target node E is encountered into queue2.

1. **QUEUE1 = {G}**
2. **QUEUE2 = {A, B, D, C, F, E}**