

Lab 12: Graph traversing

Objective:

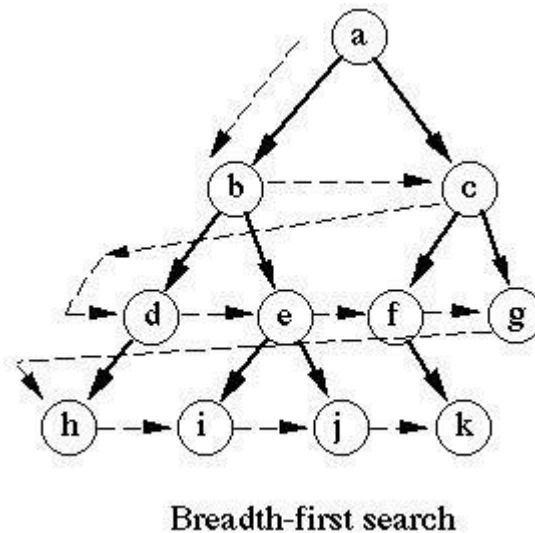
- Implementation of Depth first search for graph traversing
- Implementation of Breadth first search for graph traversing

Introduction

The breadth first search (BFS) and the depth first search (DFS) are the two algorithms used for traversing and searching a node in a graph. They can also be used to find out whether a node is reachable from a given node or not.

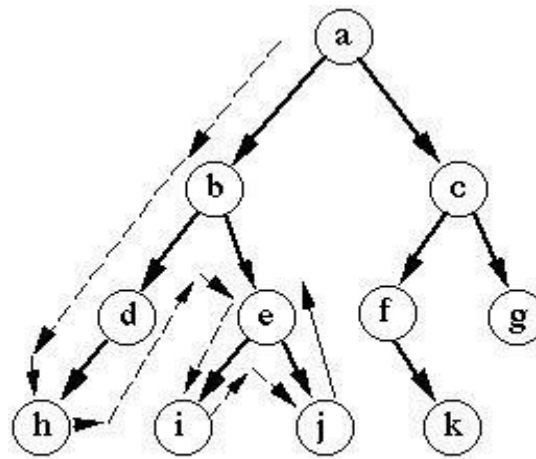
Breadth first search (BFS)

This is a very different approach for traversing the graph nodes. The aim of BFS algorithm is to traverse the graph as close as possible to the root node. Queue is used in the implementation of the breadth first search. Let's see how BFS traversal works with respect to the following graph:



Depth-first search (DFS)

A Depth-first search (DFS) algorithm begins by expanding the initial node and generating its successors. In each subsequent step, DFS expands one of the most recently generated nodes. If this node has no successors (or cannot lead to any solutions), then DFS backtracks and expands a different node. In some DFS algorithms, successors of a node are expanded in an order determined by their heuristic values. A major advantage of DFS is that its storage requirement is linear in the depth of the state space being searched.



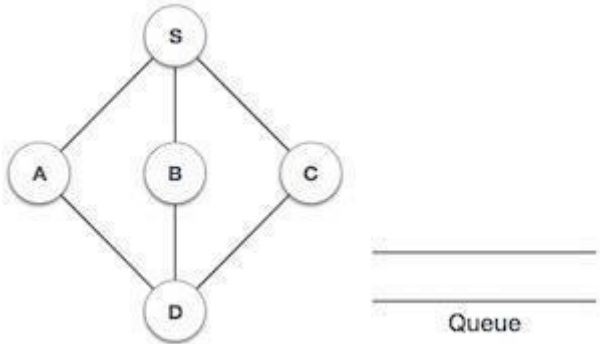
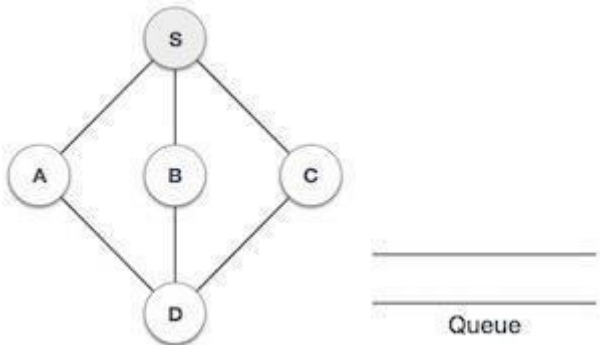
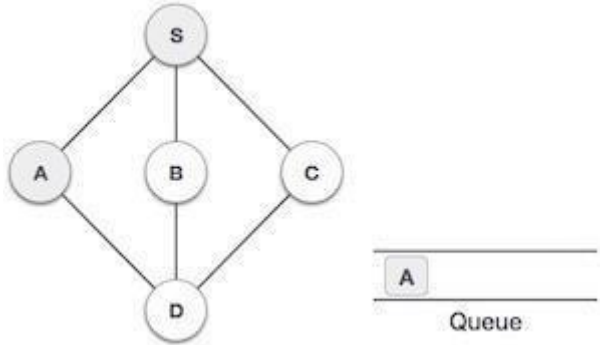
Depth-first search

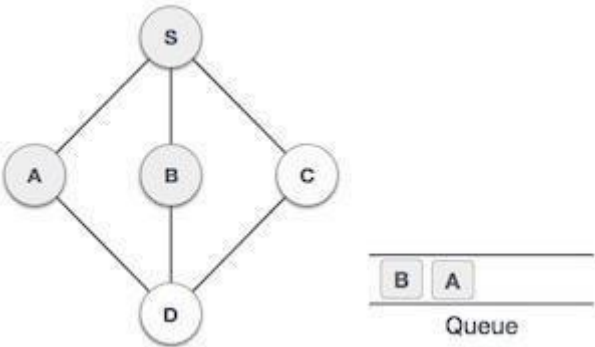
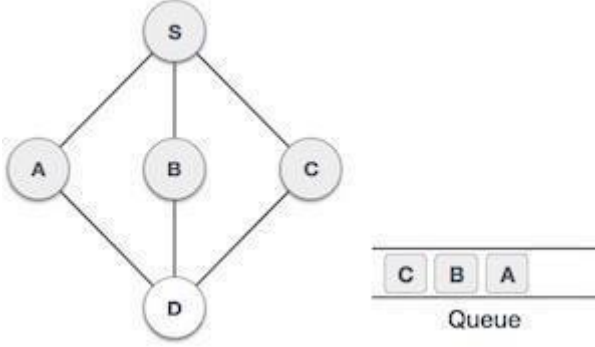
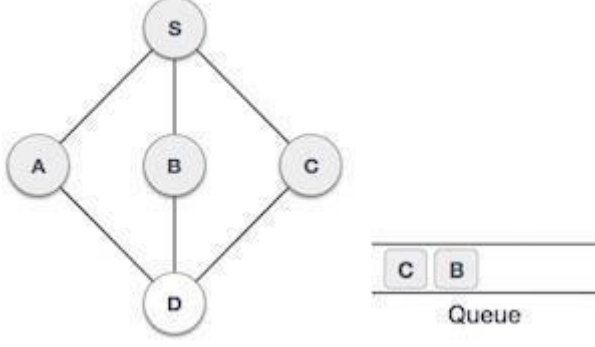
Lab Tasks:

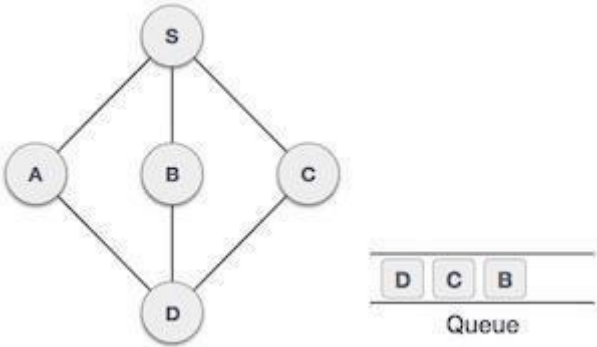
Task 1: Write a program for the Breadth First Search for the following rules.

- **Rule 1** – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
- **Rule 2** – If no adjacent vertex is found, remove the first vertex from the queue.
- **Rule 3** – Repeat Rule 1 and Rule 2 until the queue is empty.

Step	Traversal	Description

1		Initialize the queue.
2		We start from visiting S (starting node), and mark it as visited.
3		We then see an unvisited adjacent node from S . In this example, we have three nodes but alphabetically we choose A , mark it as visited and enqueue it.

4	 <p>The graph consists of five nodes: S (top), A (middle-left), B (middle-center), C (middle-right), and D (bottom). Edges connect S to A, B, and C; A to D; B to D; and C to D. A queue is shown with nodes B and A inside, labeled 'Queue' below it.</p>	<p>Next, the unvisited adjacent node from S is B. We mark it as visited and enqueue it.</p>
5	 <p>The graph is the same as in step 4. The queue now contains nodes C, B, and A, labeled 'Queue' below it.</p>	<p>Next, the unvisited adjacent node from S is C. We mark it as visited and enqueue it.</p>
6	 <p>The graph is the same as in step 5. The queue now contains nodes C and B, labeled 'Queue' below it.</p>	<p>Now, S is left with no unvisited adjacent nodes. So, we dequeue and find A.</p>

7		<p>From A we have D as unvisited adjacent node. We mark it as visited and enqueue it.</p>
---	--	---

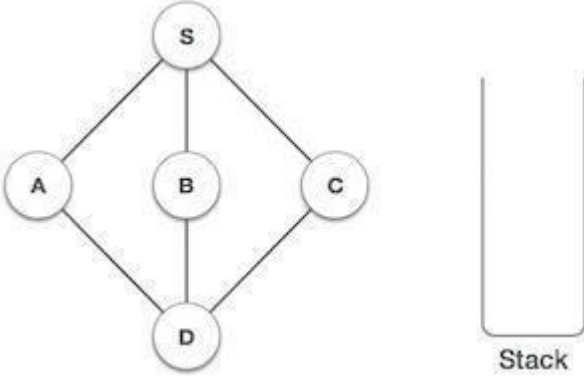
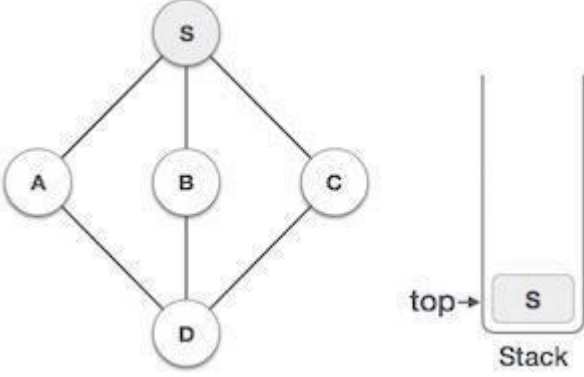
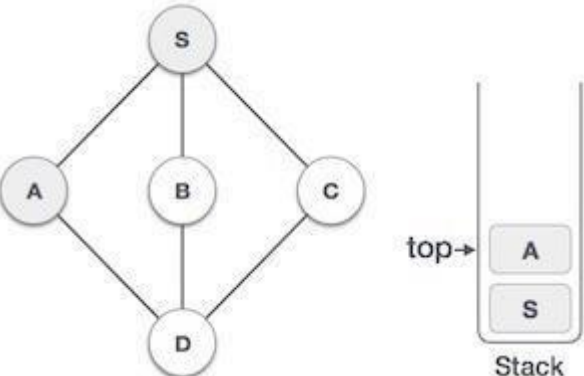
At this stage, we are left with no unmarked (unvisited) nodes. But as per the algorithm we keep on dequeuing in order to get all unvisited nodes. When the queue gets emptied, the program is over.

Task 2: Write a program for the Depth First Search for the following rules.

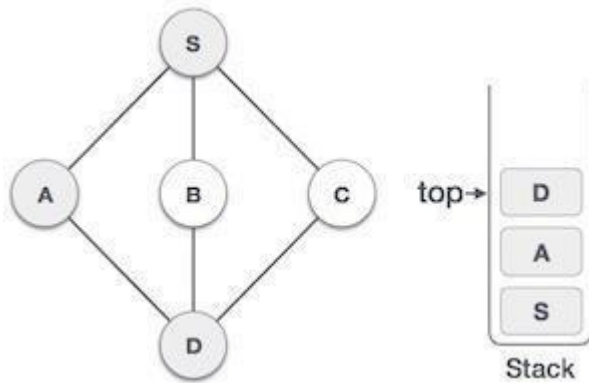
It employs the following rules.

- **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

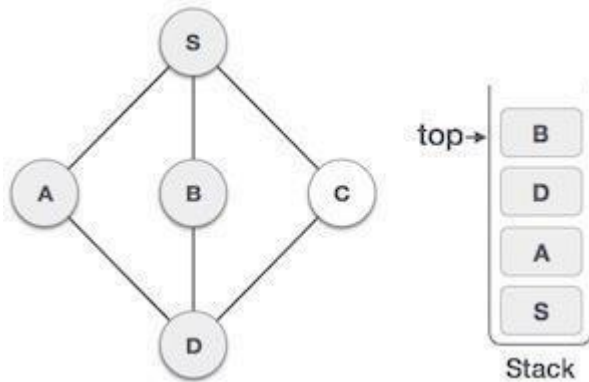
1		Initialize the stack.
2		Mark S as visited and put it onto the stack. Explore any unvisited adjacent node from S . We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order.
3		Mark A as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both S and D are adjacent to A but we are concerned for unvisited nodes only.

4

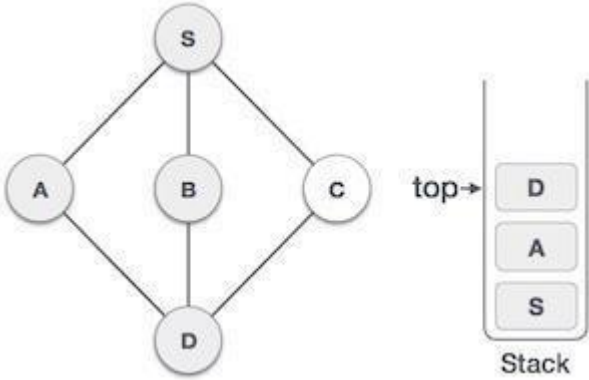
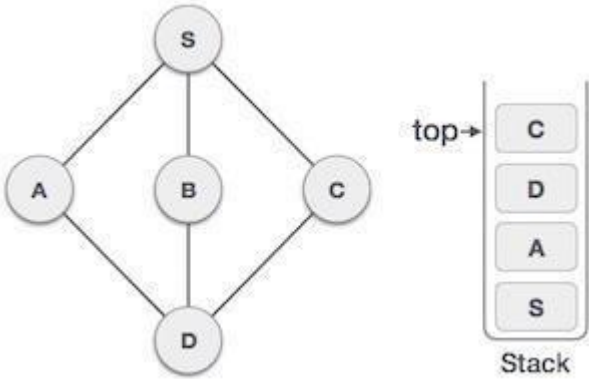


Visit **D** and mark it as visited and put onto the stack. Here, we have **B** and **C** nodes, which are adjacent to **D** and both are unvisited. However, we shall again choose in an alphabetical order.

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		<p>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.</p>
7		<p>Only unvisited adjacent node is from D is C now. So we visit C, mark it as visited and put it onto the stack.</p>

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.