



# CSCI 2270 – Data Structures

## Recitation 9

### Graph Traversal Algorithms

#### Objectives

1. What is a Graph?
2. Graph Traversals
  - a. Depth First Search (DFS)
  - b. Breadth First Search (BFS)
3. Exercises

Some aspects of the real world can be represented as a relationship between objects and the connections between them. For ex., an airline route map, the expansive road network of the United States. Modeling them helps us answer some of the questions such as

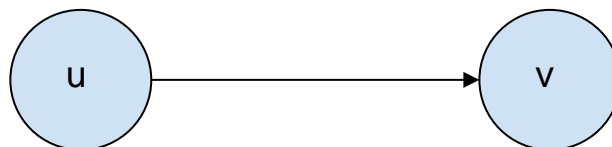
- What is the duration of a direct flight between Denver and San Francisco?
- What is the distance between Denver to San Francisco via road?

Graphs are the Data Structures used to approach these questions.

**Graph:** A graph is a pair  $(V, E)$ , where  $V$  is a set of nodes, called vertices and  $E$  is a collection of pairs of vertices, called edges.

#### Directed Edge

- Ordered pair of vertices  $(u, v)$
- First vertex  $u$  is the origin.
- Second vertex  $v$  is the destination.
- Example: one way road traffic





# CSCI 2270 – Data Structures

## Recitation 9

### Graph Traversal Algorithms

#### Undirected Edge

- Unordered pair of vertices  $(u, v)$
- Example: Railway lines



#### Applications

- Representing relationships between components in electronic circuits.
- *Transportation networks*: Highway network, Flight network
- *Computer networks*: Local area network, Internet web

#### Graph Representation

As in other Abstract Data types, to manipulate graphs we need to represent them in some useful form. Two of the most popular ways are

- Adjacency Matrix
- Adjacency Lists

#### Graph Traversals

A precursor to solving the questions that we posed above using Graphs, we must learn how to traverse them. Graph Traversal Algorithms are also called Graph Search Algorithms. The two most commonly used traversal algorithms are

- Depth First Search (DFS)
- Breadth First Search (BFS)

#### Depth First Search (DFS)

Depth first search involves choosing a vertex as a starting point, and picking one of its unvisited neighboring vertices, and doing this process recursively till all nodes are marked visited. At every step, we remember the vertex from which we visited the current vertex. This enables backtracking once we have no more neighboring vertices marked unvisited.



# CSCI 2270 – Data Structures

## Recitation 9

### Graph Traversal Algorithms

The pseudocode is shown below

```
DFS(G, u)
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)

init() {
    for each u ∈ G //u are nodes in the graph
        u.visited = false
    for each u ∈ G
        If u.visited==false
            DFS(G, u)
}
```

It is important to mark nodes as they are visited. This is done when the nodes are popped out of the Stack. And care is taken to ensure that only nodes marked as “not visited” are pushed onto the Stack. This recursive behavior (which uses a system stack) can be simulated by an iterative algorithm explicitly using a stack. So, DFS uses a stack to push nodes we mean to visit.

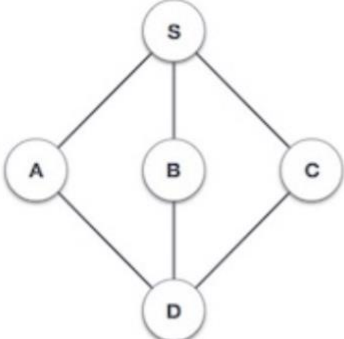

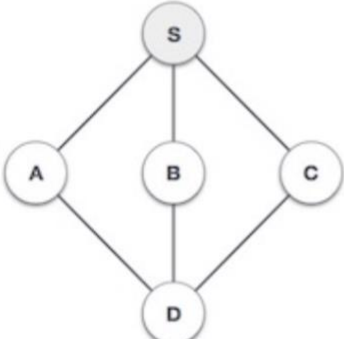

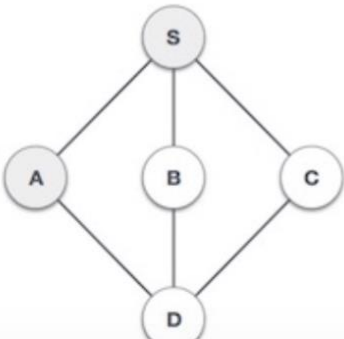
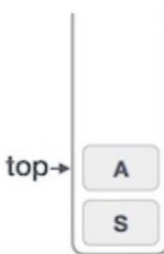
We gave a pseudocode for DFS traversal using recursion and we will be giving a visualization of DFS traversal using stacks to help you understand better. The following will give you the basic idea of Stack implementation.



# CSCI 2270 – Data Structures

## Recitation 9

### Graph Traversal Algorithms

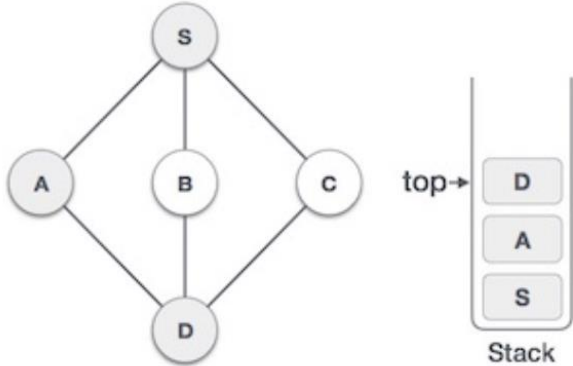
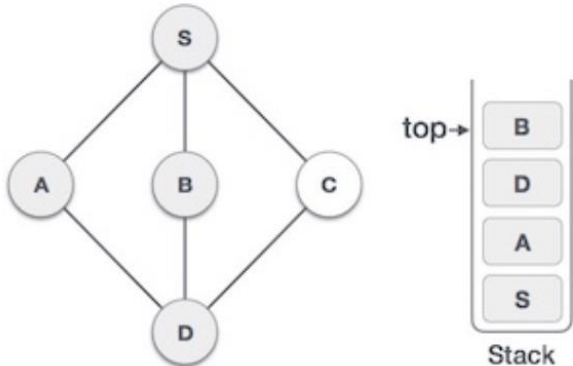
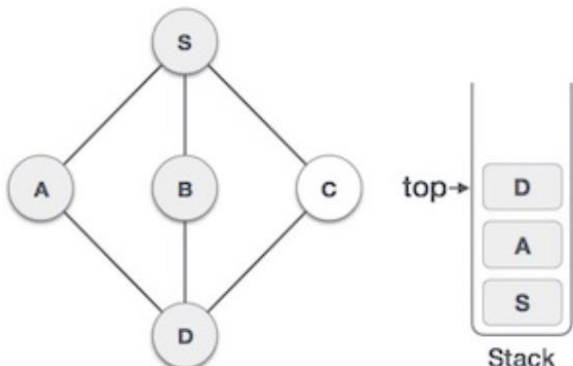
| Step | Traversal  | Description   |
|------|--|---|
| 1    |  <br>Stack     | Initialize the stack.   |
| 2    |  <br>Stack   | Mark <b>S</b> as visited and put it onto the stack. Explore any unvisited adjacent node from <b>S</b> . We have three nodes and we can pick any of them. For this example,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



# CSCI 2270 – Data Structures

## Recitation 9

### Graph Traversal Algorithms

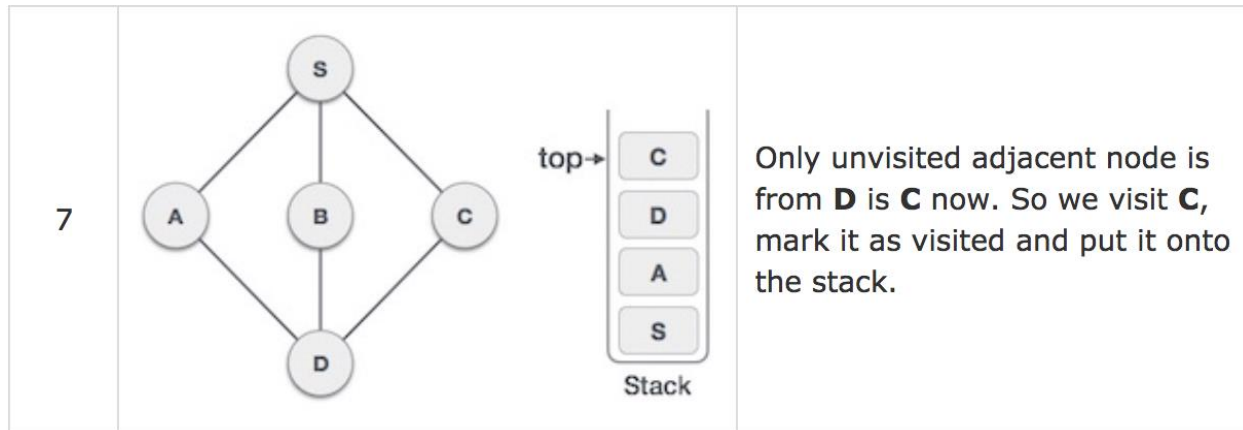
|   |   |  |
|---|---|--|
| 4 |    | Visit <b>D</b> and mark it as visited and put onto the stack. Here, we have <b>B</b> and <b>C</b> nodes, which are adjacent to <b>D</b> and both are unvisited. However, we shall again choose in an alphabetical order. |
| 5 |   | We choose <b>B</b> , mark it as visited and put onto the stack. Here <b>B</b> does not have any unvisited adjacent node. So, we pop <b>B</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 <b>D</b> to be on the top of the stack.  |



# CSCI 2270 – Data Structures

## Recitation 9

### Graph Traversal Algorithms



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 is none and we keep popping until the stack is empty.

#### Applications of DFS

- Finding connected components in an undirected graph  
(a variant of DFS is used for doing the same in directed graphs - Kosaraju algorithm)
- Solving puzzles, such as mazes (DFS helps to reach the goal faster)

#### Breadth First Search (BFS)

Breadth-first search is one of the simplest algorithms for searching a graph and the archetype for many important graph algorithms.

BFS works level by level. Initially, BFS starts at a given vertex, which is at level 0. In the first stage it visits all vertices at level 1 (its immediate neighbors). In the second stage, it visits all vertices at second level (neighbors' neighbors). These new vertices are the one which are adjacent to level 1 vertices.

BFS continues this process until all the levels of the graph are completed. Generally, Queue Data Structure is used for storing the vertices of a level. Like DFS, assume that initially all vertices are marked as unvisited. Vertices that have been processed and removed from the queue are marked visited. We use a queue to represent the visited set as it will keep the vertices in order of when they were first visited.

Breadth-first search is so named because it expands the frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier. That is, the algorithm discovers all vertices at distance  $k$  from  $s$  before discovering any vertices at distance  $k + 1$ .



# **CSCI 2270 – Data Structures**

## Recitation 9

### Graph Traversal Algorithms

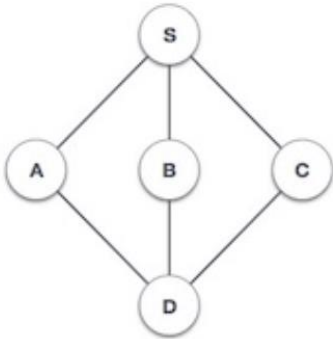
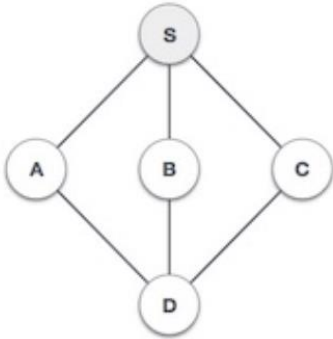
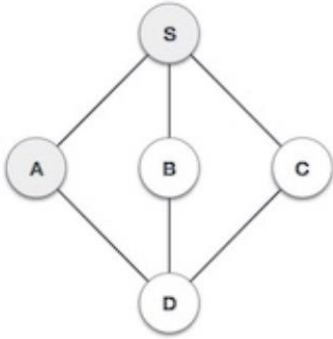
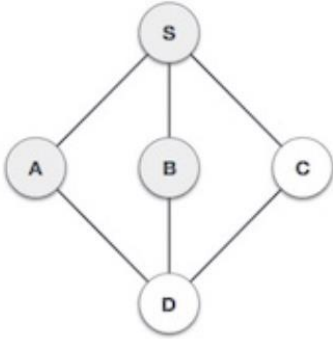
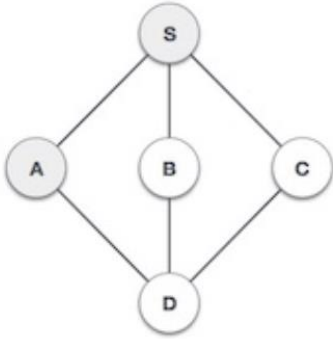
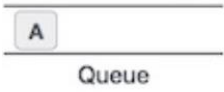
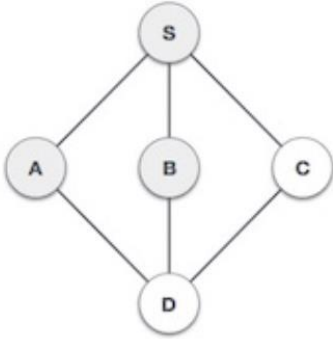
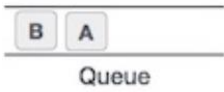
Similarly, we have an example for BFS as well using queues.



# CSCI 2270 – Data Structures

## Recitation 9

### Graph Traversal Algorithms

| Step | Traversal  | Description  |
|------|--|--|
| 1    | <br>     | Initialize the queue.  |
| 2    | <br>  | We start from visiting <b>S</b> (starting node), and mark it as visited.   |
| 3    | <br> | We then see an unvisited adjacent node from <b>S</b> . In this example, we have three nodes but alphabetically we choose <b>A</b> , mark it as visited and enqueue it. |
| 4    | <br> | Next, the unvisited adjacent node from <b>S</b> is <b>B</b> . We mark it as visited and enqueue it.  |

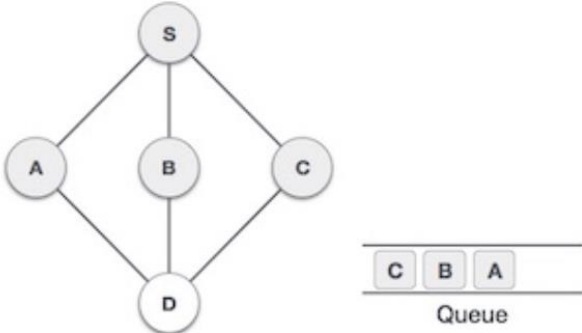
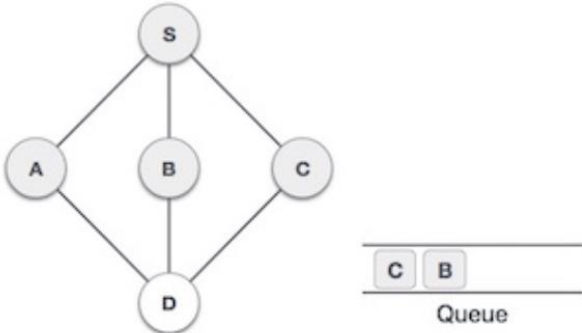
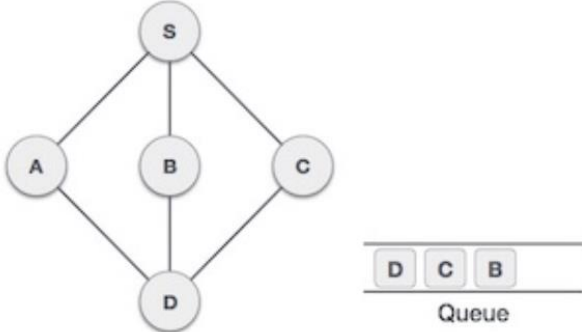




# CSCI 2270 – Data Structures

## Recitation 9

### Graph Traversal Algorithms

|   |   |   |
|---|---|---|
| 5 |    | Next, the unvisited adjacent node from <b>S</b> is <b>C</b> . We mark it as visited and enqueue it. |
| 6 |   | Now, <b>S</b> is left with no unvisited adjacent nodes. So, we dequeue and find <b>A</b> .          |
| 7 |  | From <b>A</b> we have <b>D</b> as unvisited adjacent node. We mark it as visited and enqueue it.    |

#### Applications of BFS

- Finding all connected components in an undirected graph.
- Finding the shortest path between two nodes.
- Finding nodes at distance k from a source node - e.g., find friends of friends on social media

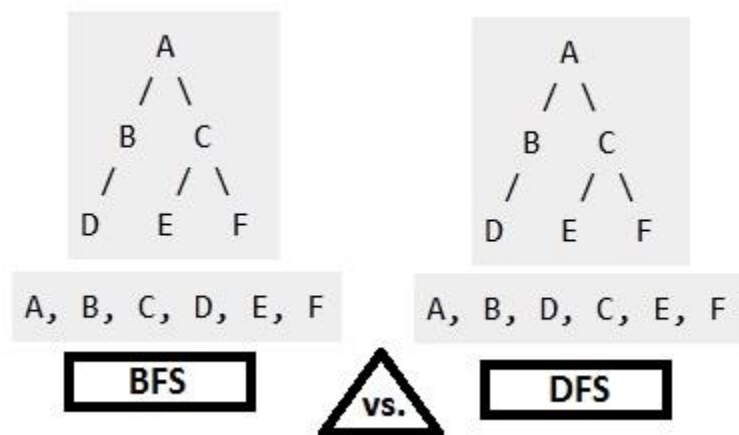


# CSCI 2270 – Data Structures

## Recitation 9

### Graph Traversal Algorithms

Comparison of BFS vs DFS on a simple graph:



## Exercise

### A. Silver Badge Problem (Mandatory)

1. Accept the assignment on Canvas, clone the repo.
2. Follow the TODOs and in Graph.cpp, complete the **findShortestPath()** function.
3. This function finds the length of the shortest path between a source and destination vertex in an undirected and unweighted graph.

### B. Gold Badge Problem

1. In Graph.cpp, complete the **printPath()** function after modifying the **findShortestPath()** function so as to print the shortest path after finding it's length.



# **CSCI 2270 – Data Structures**

## Recitation 9

### Graph Traversal Algorithms