Parul® University

Unit 6
Graph

Prof. Ashish Patel , Assistant Professor
Computer Science & Engineering

CHAPTER-6

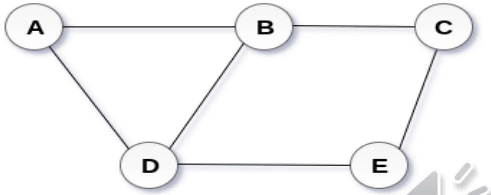
GRAPH

Basic Terminologies and Representations

Definitions: Vertices, edges, paths, etc
Representations: Adjacency list and adjacency matrix

Definitions: Graph, Vertices, Edges

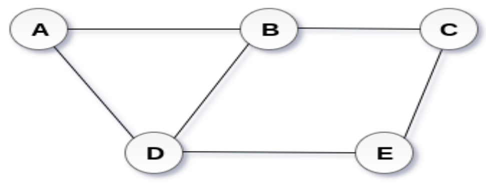
- A graph is nothing but collection of vertices and edges. Here A , B,C,D,E are the vertices and the connecting lines means edges. A graph is cyclic tree, where the vertices (Nodes) are connected with each other to represent complex relationship among them instead of having parent child relationship in tree.
- A graph G can be defined as an ordered set G(V, E) where V(G) (Read as V of G) represents the set of vertices and E(G) (Read as E of G) represents the set of edges which are used to connect these vertices.
- A Graph shown in figure represents: G(V, E) with 5 vertices (A, B, C, D, E) and six edges ((A,B), (B,C), (C,E), (E,D), (D,B), (D,A)) .



Undirected Graph

Definitions: Graph, Vertices, Edges

- Define a graph G = (V, E) :
  - V = a set of vertices (i.e. A,B,C,D,E)
  - E = a set of edges (Connection between this pairs (A,B),(A,C),(A,D),(B,D),(C,D))
- Example:
  - V = {A,B,C,D}
  - E = {(A,B),(A,C),(A,D),(B,D),(C,D)}

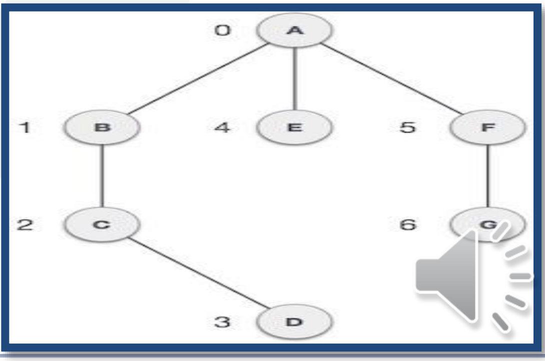


Undirected Graph

Graph data Structure

•**Vertex** – Each node (i.e. Circle with name) of the graph is represented as a vertex. The labelled circle represents vertices (i.e. A to G are vertices). We can represent them using an array with index 0 to n-1 vertices. Here A can be identified with index 0. B can be identified with index 1 and so on.

•**Edge** – Edge represents is a connection line between two vertices or link between two vertices. In the figure the lines or link from A to B, B to C, C to D and so on represents edges. We can use a two-dimensional array to represent a graph in memory (i.e. AB can be represented as 1 at row 0, column 1, BC as 1 at row 1, column 2, AF as 1 at row 0, column 5 and so on, keeping other combinations as 0).



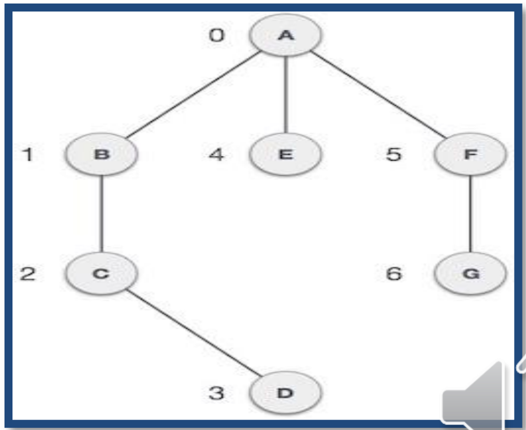
Parul® University

DIGITAL LEARNING CONTENT

Graph data Structure

Adjacency – Two node or vertices are connected means they are adjacent to each other with edge. In the following example, B is adjacent to A, E is adjacent to A, B is adjacent to C, and so on.

Path – Path represents a sequence of connection between the two vertices. In the following example, ABCD represents a path from A to D and AFG represents path from A to G like that



Basic Operations

Add Vertex or node in a graph .

Add Edge to connect two vertices or nodes of the graph.

Display Vertex or node of the graph.

Graph Classifications

There are several common kinds of graphs

- Weighted or Un weighted
- Directed or Undirected
- Cyclic or Acyclic

Weighted Graph & Unweighted Graph

Weighted graph:

- Edge showing with some numbers ,Weight typically shows cost of traversing. Example: weights are distances between to cities.

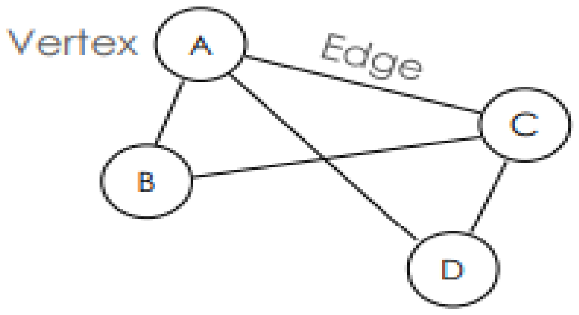
Un weighted graph:

- Edges have no weight
- Edges simply show connections. Example: Line between two city without any information.

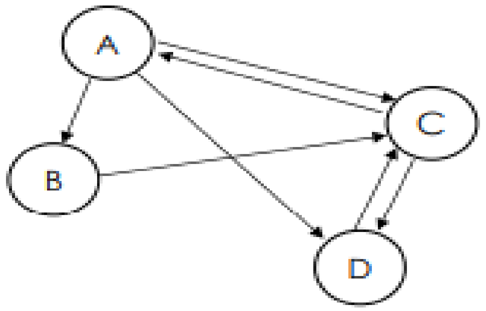
Directed & Undirected

•**Undirected Graphs**: each edge can be traversed in **either direction(**i.e. not having any direction)

•**Directed Graphs**: each edge can be traversed **only in a specified direction** shown by arrow.



Undirected



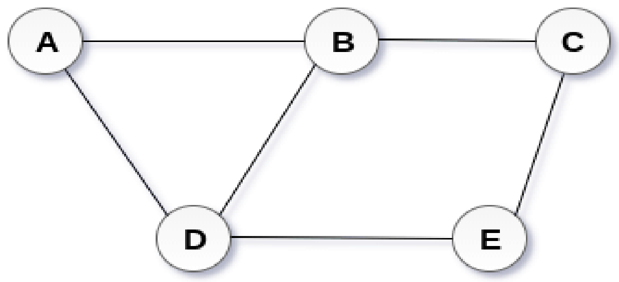
Directed

Image source : Google

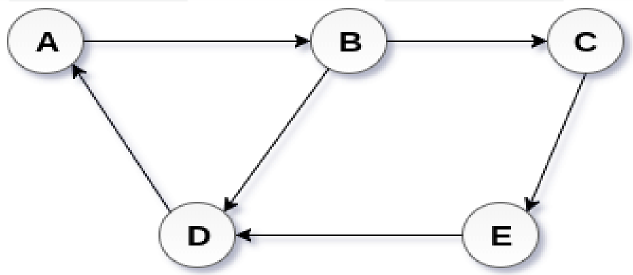
Directed & Undirected

•In an **undirected graph**, edges are not associated with the directions with them. If an edge exists between vertex A and B then the vertices can be traversed from B to A as well as A to B.

•In a **directed graph**, edges form an ordered pair. Edges represent a specific path from some node A to another node B. Node A is called initial node (I.e. Source) while node B is called terminal node(I.e. Destination).



Undirected Graph



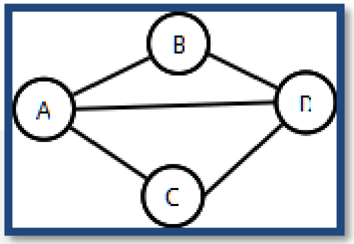
Directed Graph

Image source : Google

Cyclic & Acyclic

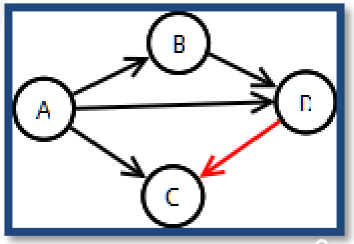
- A **Cyclic** graph contains cycles

  Example: roads (normally)



- An **acyclic** graph contains no cycles

  Example: roads (Directions)



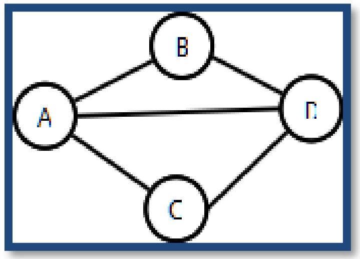
Data Structures for Representing Graphs

Two common data structures for representing graphs:

- Adjacency lists
- Adjacency matrix

Graphs: Terminology Involving Paths

Path: sequence of vertices in which vertices are connect from initial node to terminal node is connected by an edge.
Cycle: a path that starts and ends on the same vertex or node.
Simple path: No vertex is repeated (except first and last)
    Simple paths cannot contain cycles
Length of a path: Number of edges in the path or sometimes the sum of the weights of the edges
Examples
    A sequence of vertices: (A, B, C, D) [Is this path, simple path, cycle?]
    (A, B, D, A, C) [path, simple path, cycle?]
    Cycle: ?
    Lengths?

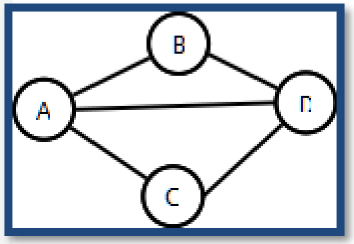


Adjacency List Representation

Each node has a list of adjacent nodes:

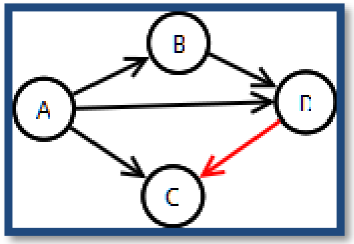
Example (undirected graph):

A: B, C, D
B: A, D
C: A, D
D: A, B, C



Example (directed graph):

A: B, C, D
B: D
C: Nil
D: C



Adjacency List Representation

Weighted graph can store weights in list

Space: $\Theta(V + E)$ (ie $|V| + |E|$)

Time:

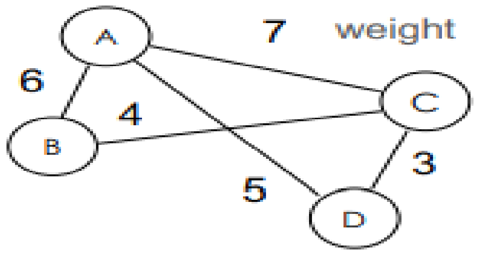
To visit each node that is adjacent to node u: $\Theta(\text{degree}(u))$

To determine if node u is adjacent to node v: $\Theta(\text{degree}(u))$

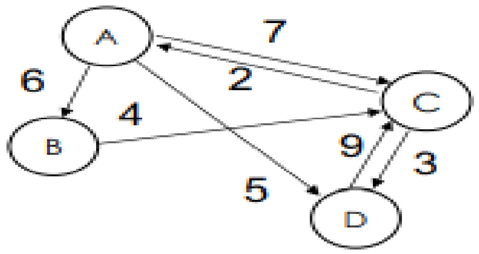
Adjacency Matrix Representation

Adjacency Matrix: 2D array containing weights on edges as shown in figure.

- Contain weight of edge from row vertex to column vertex
- Contain ∞ last if no edge from row vertex to column vertex
- Contain 0 on diagonal (if self edges not allowed)



from \ to	A	B	C	D
A	0	6	7	5
B	6	0	4	∞
C	7	4	0	3
D	5	∞	3	0



from \ to	A	B	C	D
A	0	6	7	5
B	∞	0	4	∞
C	2	∞	0	3
D	∞	∞	9	0

Graphs Search and Traversal Algorithms

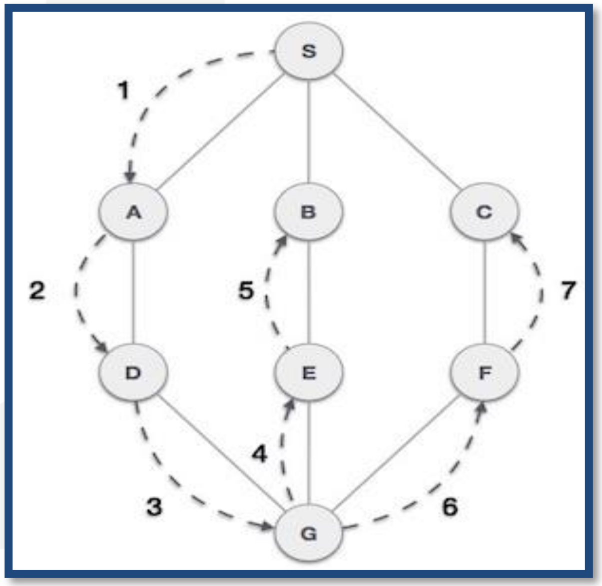
Depth First Search (DFS) algorithm traverses a graph in a depth ward (From Bottom) motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

DFS algorithm traverses from S to A to D to G to E to B first, then to F and lastly to C. It apply following rules.

Rule 1 – Visit the adjacent unvisited vertex or node. Mark it as visited. Display it. Push it in a stack.

Rule 2 – If no adjacent vertex or node 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.)

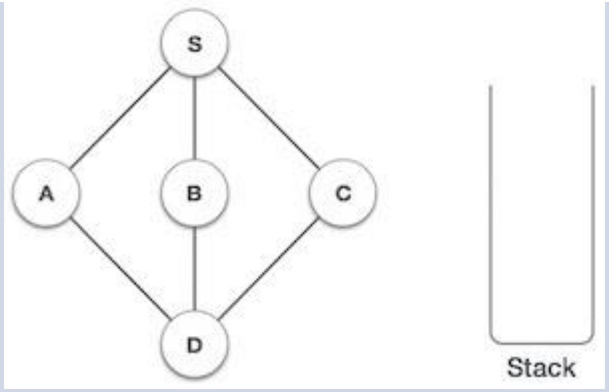
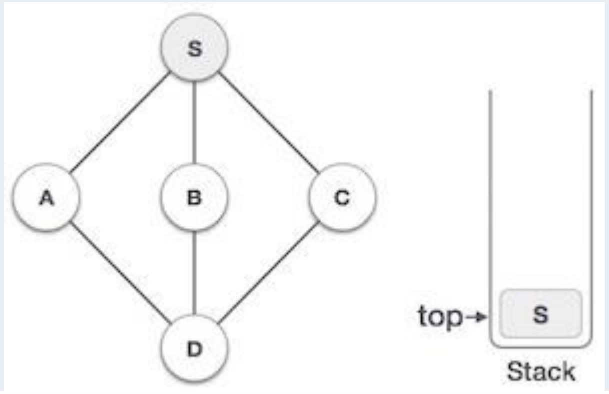
Rule 3 – Repeat Rule 1 and Rule 2 until the stack is empty.



Parul® University

DIGITAL LEARNING CONTENT

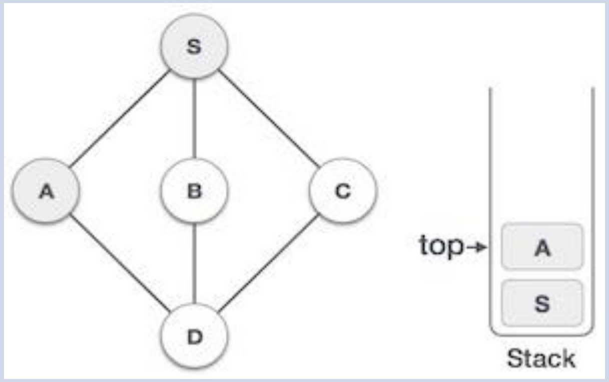
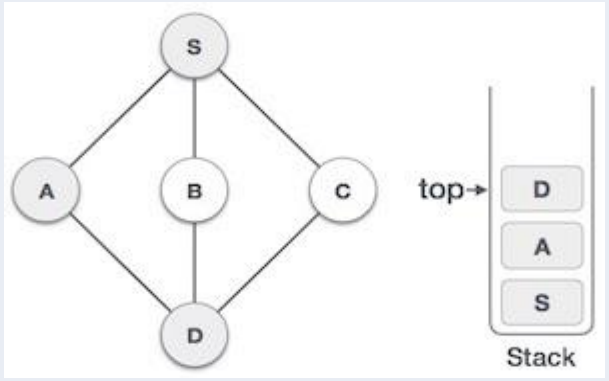
Depth First Search(DFS) Algorithm Traverses

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.

Parul® University

DIGITAL LEARNING CONTENT

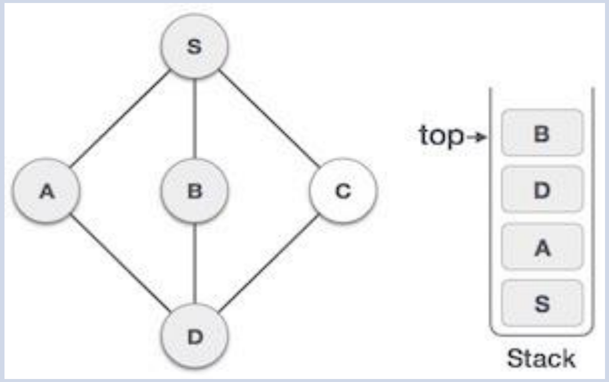
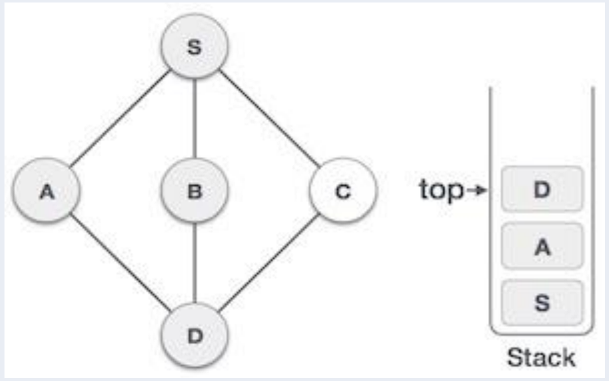
Depth First Search(DFS) Algorithm Traverses

Step	Traversal	Description
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.

Parul® University

DIGITAL LEARNING CONTENT

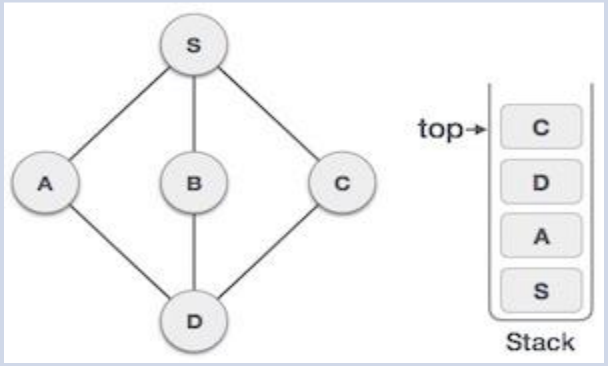
Depth First Search(DFS) Algorithm Traverses

Step	Traversal	Description
5		Initialize 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 **D** to be on the top of the stack.

Parul® University

DIGITAL LEARNING CONTENT

Graphs Search and Traversal Algorithms

Step	Traversal	Description
7		Only unvisited adjacent node is from **D** is **C** now. So we visit **C**, mark it as visited and put it onto the stack.

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.

Breadth First Search(BFS) Algorithm Traverses

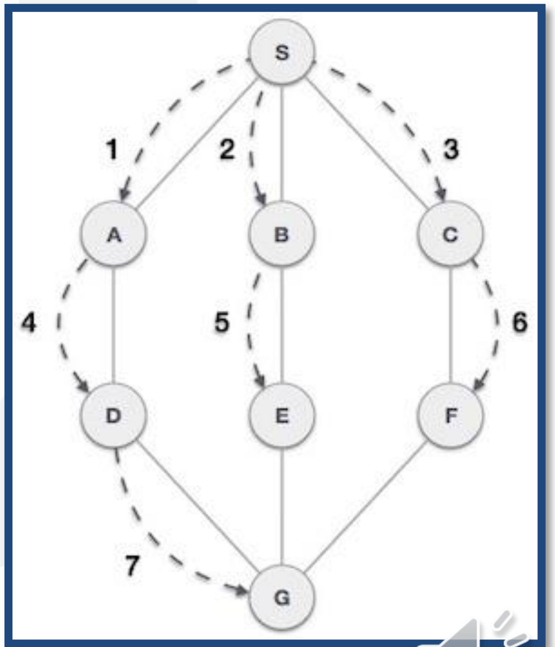
Breadth First Search (BFS) algorithm traverses a graph in a breadth ward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It apply 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.



Question

- Q.1. What are the applications of DFS Algorithms?

Answer Keywords-

  - Minimum spanning tree
  - Detecting cycle in a graph
  - Path Finding
  - <u>Topological Sorting</u>
  - Solving puzzles with only one solution

- Q.2. Detect cycle in an undirected graph

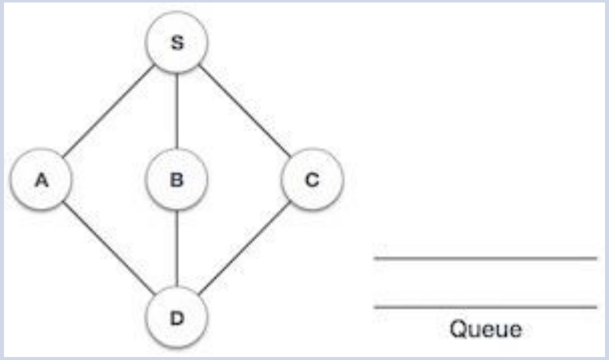
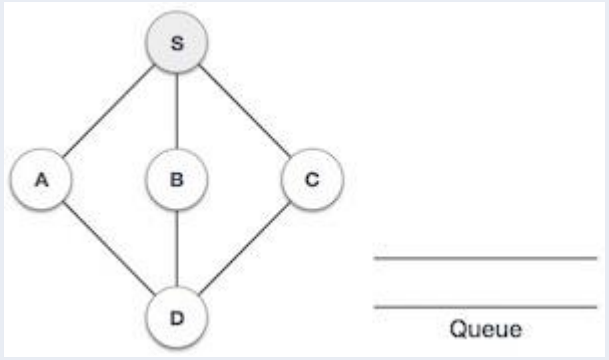
Answer Keywords-

  - Travasal with DFS
  - Recursion
  - Stack

Parul® University

DIGITAL LEARNING CONTENT

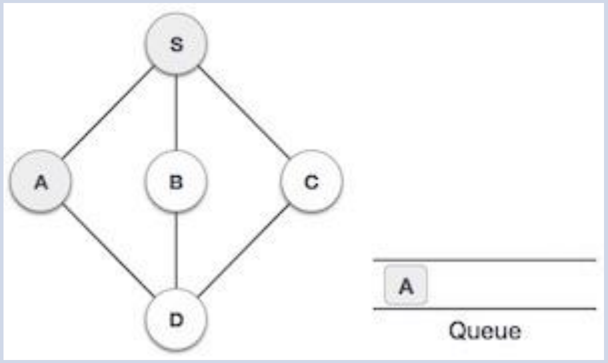
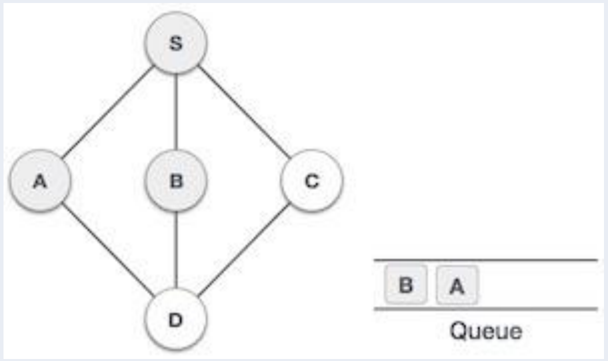
Breadth First Search(BFS) Algorithm Traverses

Step	Traversal	Description
1		Initialize the queue.
2		We start from visiting **S** (starting node), and mark it as visited.

Parul® University

DIGITAL LEARNING CONTENT

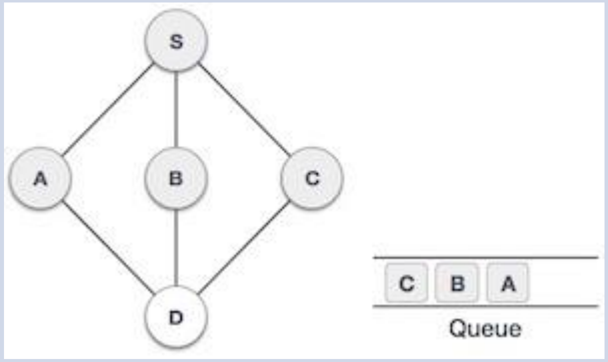
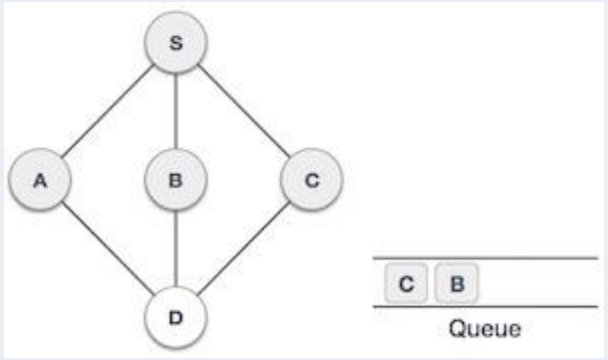
Breadth First Search(BFS) Algorithm Traverses

Step	Traversal	Description
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		Next, the unvisited adjacent node from **S** is **B**. We mark it as visited and enqueue it.

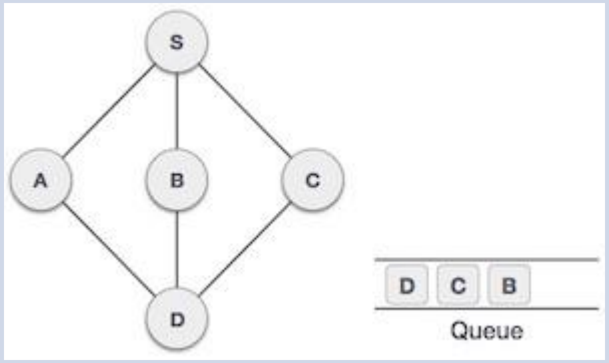
Parul® University

DIGITAL LEARNING CONTENT

Breadth First Search(BFS) Algorithm Traverses

Step	Traversal	Description
5		Next, the unvisited adjacent node from **S** is **C**. We mark it as visited and enqueue it.
6		Now, **S** is left with no unvisited adjacent nodes. So, we dequeue and find **A**.

Breadth First Search(BFS) Algorithm Traverses

Step	Traversal	Description
7		From **A** we have **D** as unvisited adjacent node. We mark it as visited and enqueue it.

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.

Parul® University

DIGITAL LEARNING CONTENT

Breadth First Search(BFS) Complexity

Assume that graph is connected. Depth-first search visits every vertex in the graph and checks every edge its edge. Therefore, DFS complexity is O(V + E). As it was mentioned before, if an adjacency matrix is used for a graph representation, then all edges, adjacent to a vertex can‘t be found efficiently, that results in O(V2) complexity.

Pre Requisites

As with the depth first search (DFS), the discovery edges form a spanning tree, which in this case we call the BSF-tree.

BSF used to solve following problem
•Testing whether graph is connected.
•Computing a spanning forest of graph.
•Computing, for every vertex in graph, a path with the minimum number of edges between
•Start vertex and current vertex or reporting that no such path exists.
• Computing a cycle in graph or reporting that no such cycle exists.

Analysis
Total running time of BFS is O(V + E).

Depth First Search(DFS) Complexity

As with the depth first search (DFS), the discovery edges form a spanning tree, which in this case we call the BSF-tree.

BSF used to solve following problem

- Testing whether graph is connected.
- Computing a spanning forest of graph.
- Computing, for every vertex in graph, a path with the minimum number of edges between start vertex and current vertex or reporting that no such path exists.
- Computing a cycle in graph or reporting that no such cycle exists.

Analysis

Total running time of BFS is O(V + E).

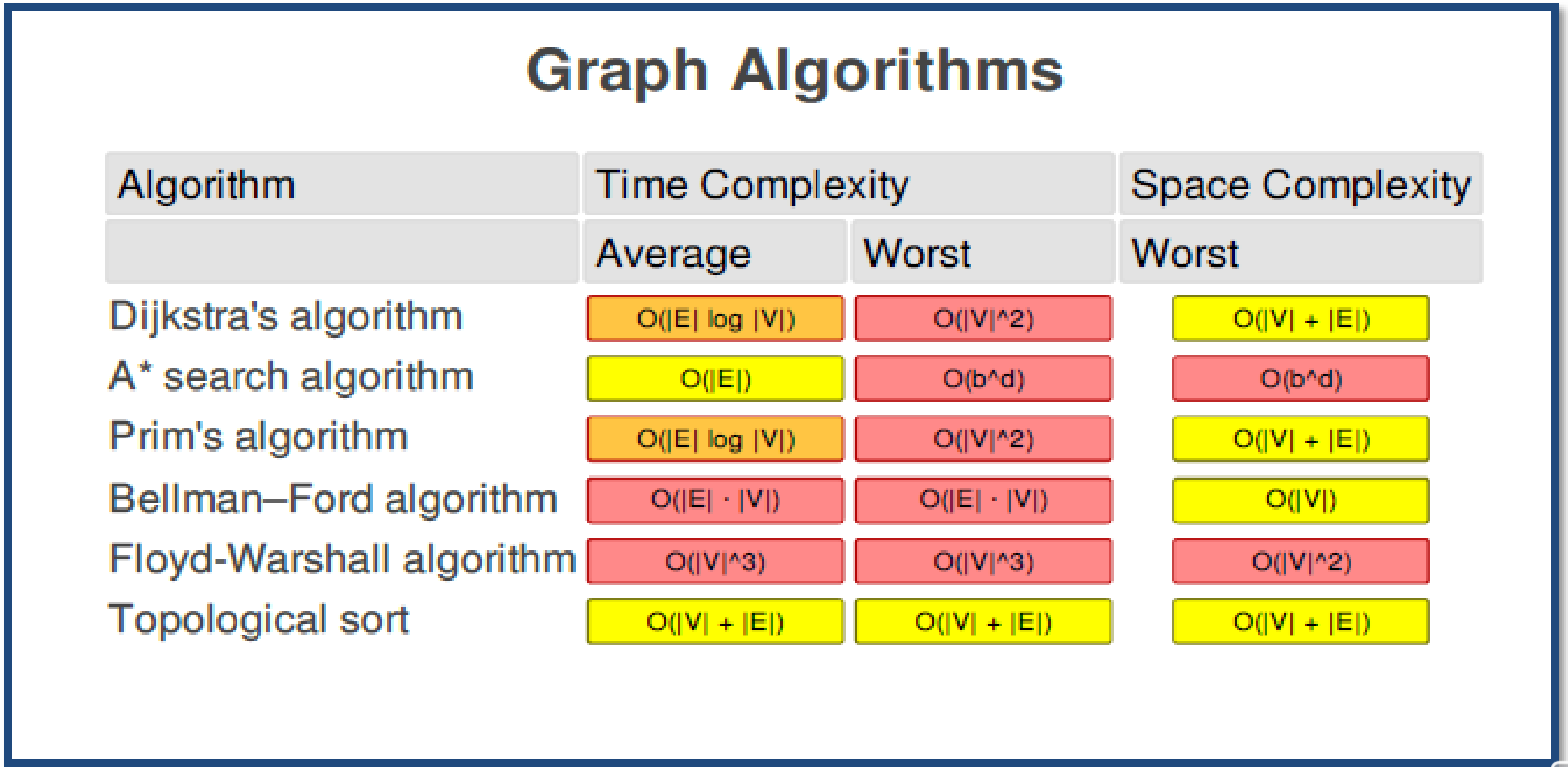
Question

- Q.1. What are the applications of BFS Algorithms?

Answer Keywords-

- – Flood Fill Algorithms
- – Count the number of islands.
- – Find shortest safe route in a field with sensors present.
- – Minimum Spanning Tree

Complexity



Graph Algorithms

Algorithm	Time Complexity		Space Complexity
	Average	Worst	Worst
Dijkstra's algorithm	O(|E| log |V|)	O(|V|^2)	O(|V| + |E|)
A* search algorithm	O(|E|)	O(b^d)	O(b^d)
Prim's algorithm	O(|E| log |V|)	O(|V|^2)	O(|V| + |E|)
Bellman–Ford algorithm	O(|E| · |V|)	O(|E| · |V|)	O(|V|)
Floyd-Warshall algorithm	O(|V|^3)	O(|V|^3)	O(|V|^2)
Topological sort	O(|V| + |E|)	O(|V| + |E|)	O(|V| + |E|)

Image source : Google

Applications of Graphs

- Finding Minimum Spanning Trees (MST) – Prim's and Kruskal's Algorithm
- Shortest path - Dijkstra's Algorithm
- Transitive closure – Warshall's Algorithm
- Topological sort.

Applications of Graphs

•**Social network graphs:** Graphs are used to represent who knows whom, who communicates with whom, who influences whom or other relationships in social structures. For Example in twitter graph of who follows whom. These can be used to determine how information flows, how topics become hot, how communities develop, or even who might be a good match for who, or is that whom.

•**Transportation networks.** In road networks vertices are intersections and edges are the road segments between them, and for public transportation networks vertices are stops and edges are the links between them. Such networks are used by many map programs such as Google maps, Bing maps and now Apple IOS 6 maps (well perhaps without the public transport) to find the best routes between locations. They are also used for studying traffic patterns, traffic light timings, and many aspects of transportation.

Applications of Graphs

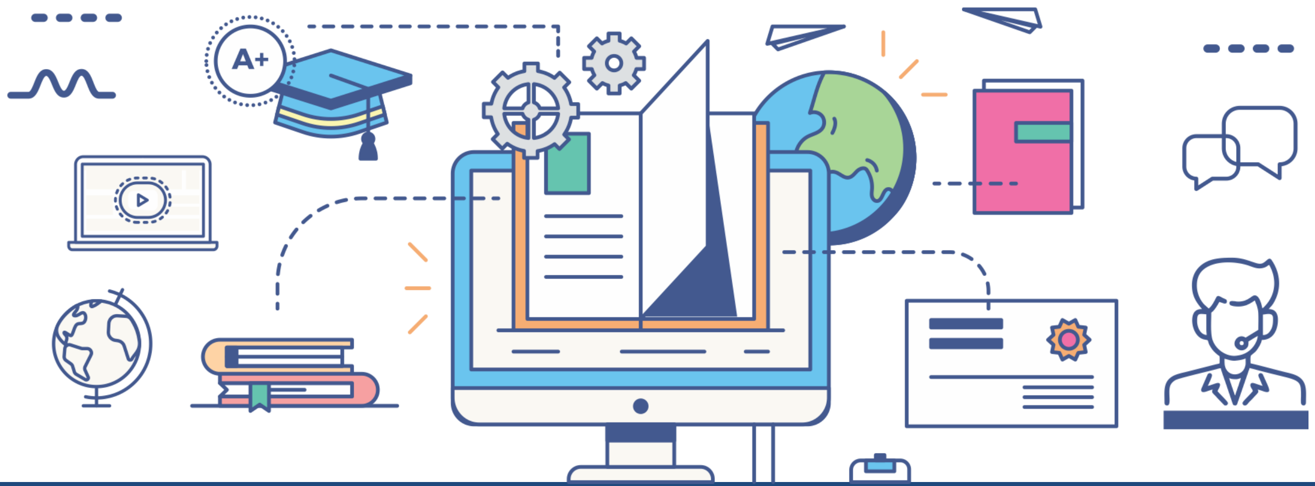
- **Utility graphs.** The power grid, the Internet, and the water network are all examples of graphs where vertices represent connection points, and edges the wires or pipes between them. Analyzing properties of these graphs is very important in understanding the reliability of such utilities under failure or attack, or in minimizing the costs to build infrastructure that matches required demands.
- **Document link graphs.** The best known example is the link graph of the web, where each web page is a vertex, and each hyperlink a directed edge. Link graphs are used, for example, to analyze relevance of web pages, the best sources of information, and good link sites.
- **Network packet traffic graphs.** Vertices are IP (Internet protocol) addresses and edges are the packets that flow between them. Such graphs are used for analyzing network security, studying the spread of worms, and tracking criminal or non-criminal activity.

Applications of Graphs

- **Protein-protein interactions graphs**. Vertices represent proteins and edges represent interactions between them that carry out some biological function in the cell. These graphs can be used, for example, to study molecular pathways—chains of molecular interactions in a cellular process. Humans have over 120K proteins with millions of interactions among them.

- **Neural networks.** Vertices represent neurons and edges the synapses between them. Neural networks are used to understand how our brain works and how connections change when we learn.

DIGITAL LEARNING CONTENT



Parul® University

www.paruluniversity.ac.in