Data Structure

Dr. Ghanshyam Rathod, Assistant Professor
IT and Computer Science

Parul® University

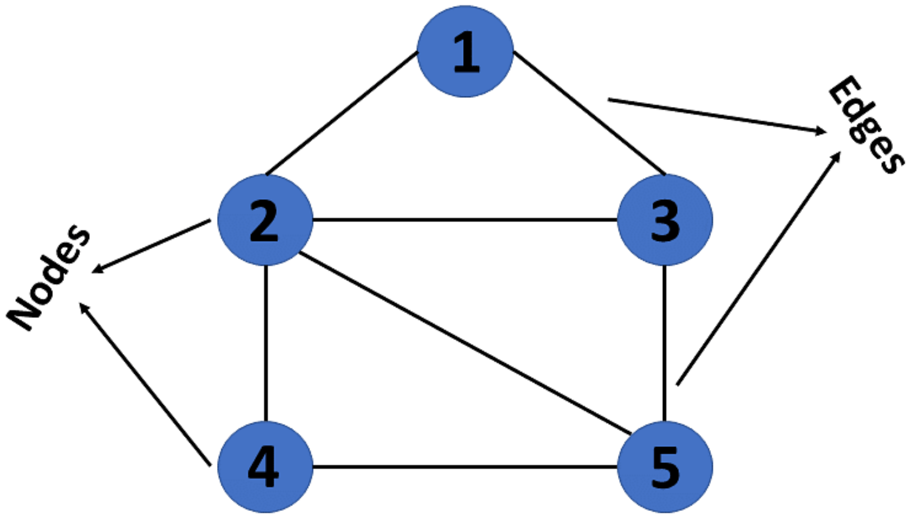
DIGITAL LEARNING CONTENT

CHAPTER-5

Graph

Graph

- A graph is a non-linear kind of data structure made up of nodes or vertices and edges. The edges connect any two nodes in the graph, and the nodes are also known as vertices.



- This graph has a set of vertices V= { 1,2,3,4,5} and
- A set of edges E= { (1,2),(1,3),(2,3),(2,4),(2,5),(3,5),(4,5) }.

Graph Terminologies

1. **Graph:** A Graph G is a non-empty set of vertices (or nodes) V and a set of edges E, where each edge connects a pair of vertices. Formally, a graph can be represented as G= (V, E). Graphs can be classified based on various properties, such as directedness of edges and connectivity.

2. **Vertex (Node):** A Vertex, often referred to as a Node, is a fundamental unit of a graph. It represents an entity within the graph. In applications like social networks, vertices can represent individuals, while in road networks, they can represent intersections or locations.

3. **Edge:** An Edge is a connection between two vertices in a graph. It can be either directed or undirected. In a directed graph, edges have a specific direction, indicating a one-way connection between vertices. In contrast, undirected graphs have edges that do not have a direction and represent bidirectional connections.

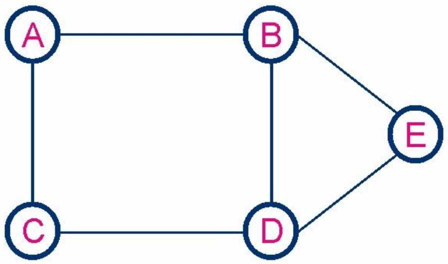
Graph Terminologies

4. **Degree of a Vertex:** The Degree of a Vertex in a graph is the number of edges incident to that vertex. In a directed graph, the degree is further categorized into the in-degree (number of incoming edges) and out-degree (number of outgoing edges) of the vertex.

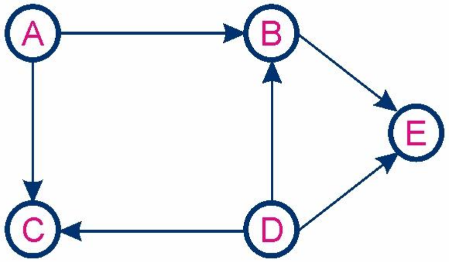
5. **Path:** A Path in a graph is a sequence of vertices where each adjacent pair is connected by an edge. Paths can be of varying lengths and may or may not visit the same vertex more than once.

6. **Loop:** An edge that is associated with the similar end points can be called as Loop.

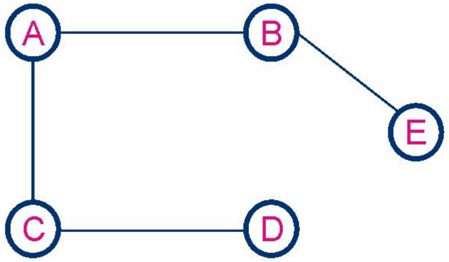
7. **Adjacent Nodes:** If two nodes u and v are connected via an edge e, then the nodes u and v are called as neighbors or adjacent nodes.



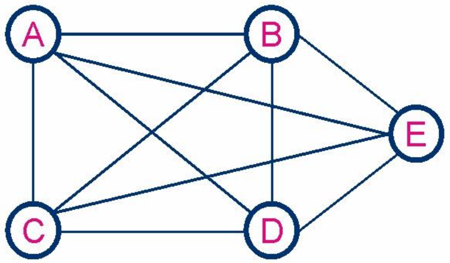
Undirected Graph



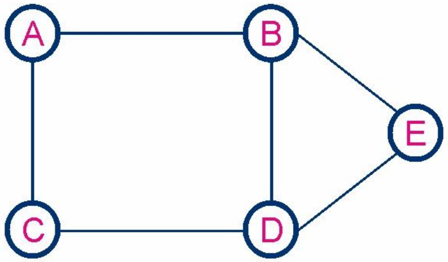
Directed Graph



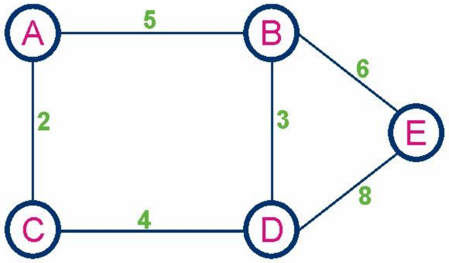
Sparce Graph



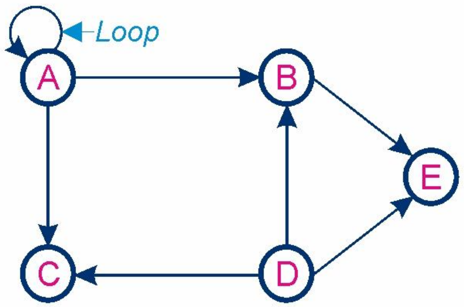
Complete Graph (Dense)



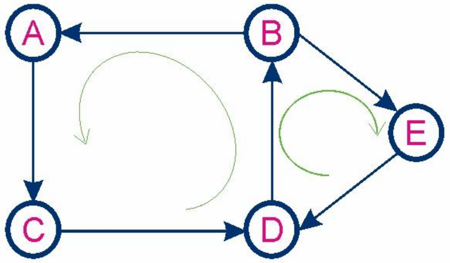
Unweighted Graph



Weighted Graph



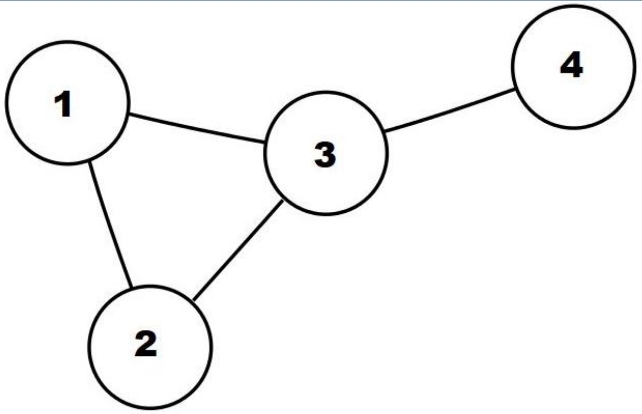
Acyclic Graph



Cyclic Graph

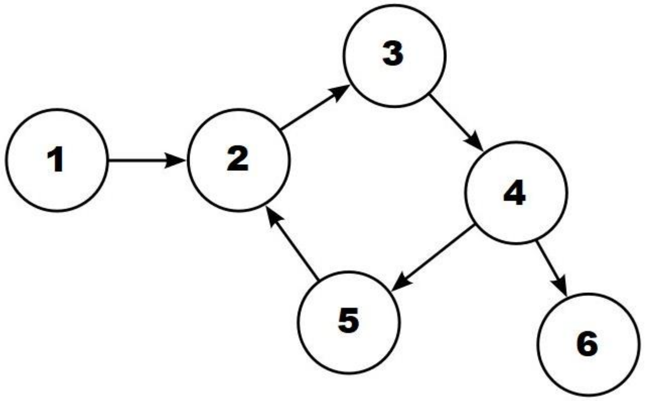
Types of Graph

1. **Undirected:** A graph in which all the edges are bi-directional. The edges do not point in a specific direction.



Undirected Graph

2. **Directed:** A graph in which all the edges are uni-directional. The edges point in a single direction.



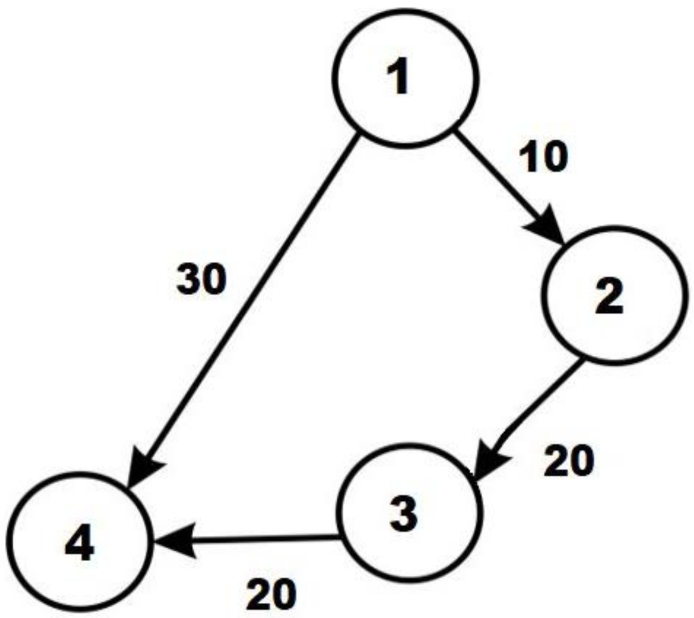
Directed Graph

Types of Graph

3. **Weighted Graph:** A graph with a value associated with every edge. The values corresponding to the edges are called weights. A value in a weighted graph can represent quantities such as cost, distance, and time, depending on the graph. We typically use weighted graphs in modelling computer networks.

   An edge in a weighted graph is represented as (u, v, w), where:

   - u is the source vertex
   - v is the destination vertex
   - w represents the weight associated with going from u to v



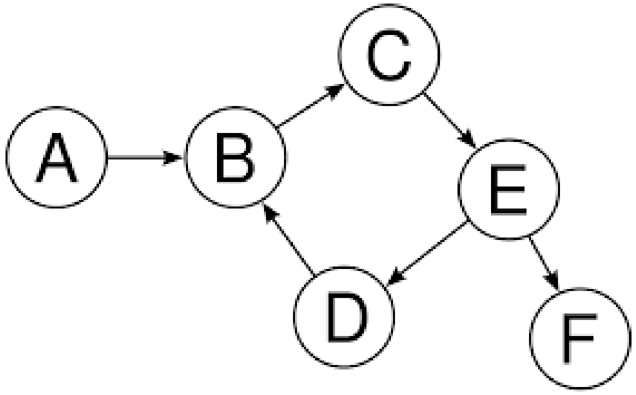
Weighted Graph

Types of Graph

4. **Cyclic Graph:** A cyclic graph is a directed graph which contains a path from at least one node back to itself. In simple terms cyclic graphs contain a cycle.

   There exists a path from node B which connects it to itself. The path is {(B,C),(C,E),(E,D),(D,B)}.

   Similarly there also exists a path from node D back to itself. The path is {(D,B) (B,C) (C,E) (E,D)}

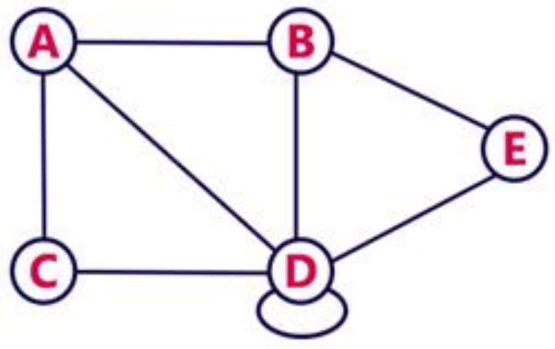


Representation of Graph (Using Adjacency Matrix)

In this representation, graph can be represented using a matrix of size total number of vertices by total number of vertices 5 * 5 (5 rows and 5 columns).

That means if a graph with 4 vertices can be represented using a matrix of 4X4 matrix. In this matrix, rows and columns both represents vertices.

This matrix is filled with either 1 or 0. Here, 1 represents there is an edge from row vertex to column vertex and 0 represents there is no edge from row vertex to column vertex.

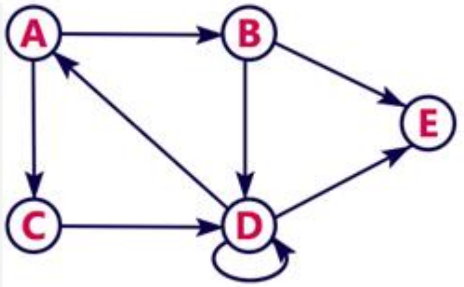


Un-Directed Graph

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 1 & 1 & 0 \\ B & 1 & 0 & 0 & 1 & 1 \\ C & 1 & 0 & 0 & 1 & 0 \\ D & 1 & 1 & 1 & 1 & 1 \\ E & 0 & 1 & 0 & 1 & 0 \end{array}$$

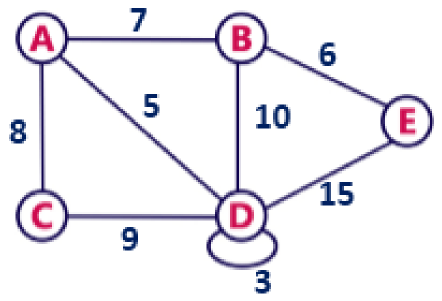
Representation of Graph (Using Adjacency Matrix)

In the this directed graph, 1 represents an edge from row vertex to column vertex, and 0 represents no edge from row vertex to column vertex.



$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 1 & 0 & 0 \\ B & 0 & 0 & 0 & 1 & 1 \\ C & 0 & 0 & 0 & 1 & 0 \\ D & 1 & 0 & 0 & 1 & 1 \\ E & 0 & 0 & 0 & 0 & 0 \end{array}$$

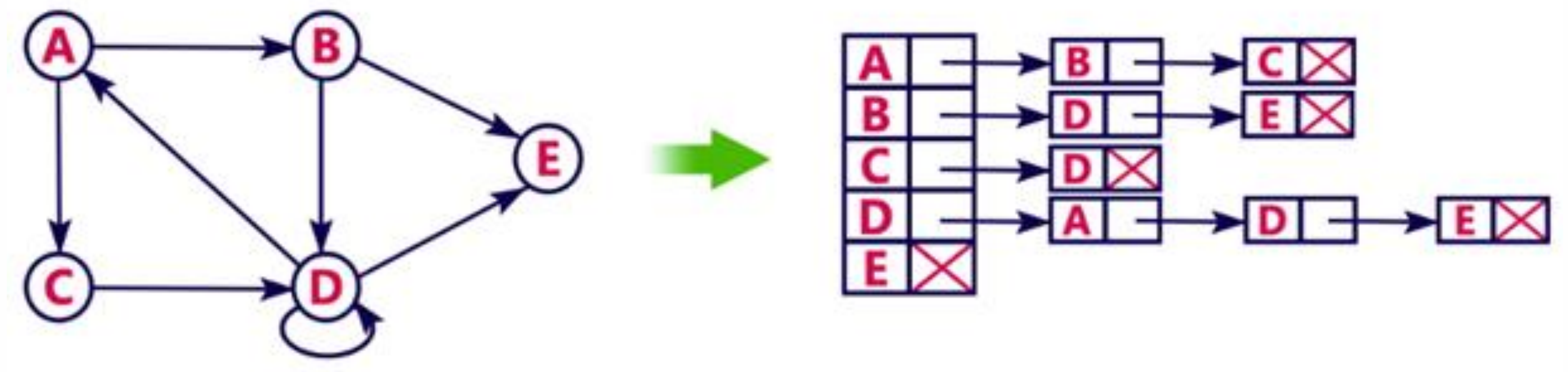
In this undirected weighted graph, weights are stored instead of 1.



$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 7 & 8 & 5 & 0 \\ B & 7 & 0 & 0 & 10 & 6 \\ C & 8 & 0 & 0 & 9 & 0 \\ D & 5 & 10 & 9 & 3 & 15 \\ E & 0 & 6 & 0 & 15 & 0 \end{array}$$

Representation of Graph (Using Adjacency List)

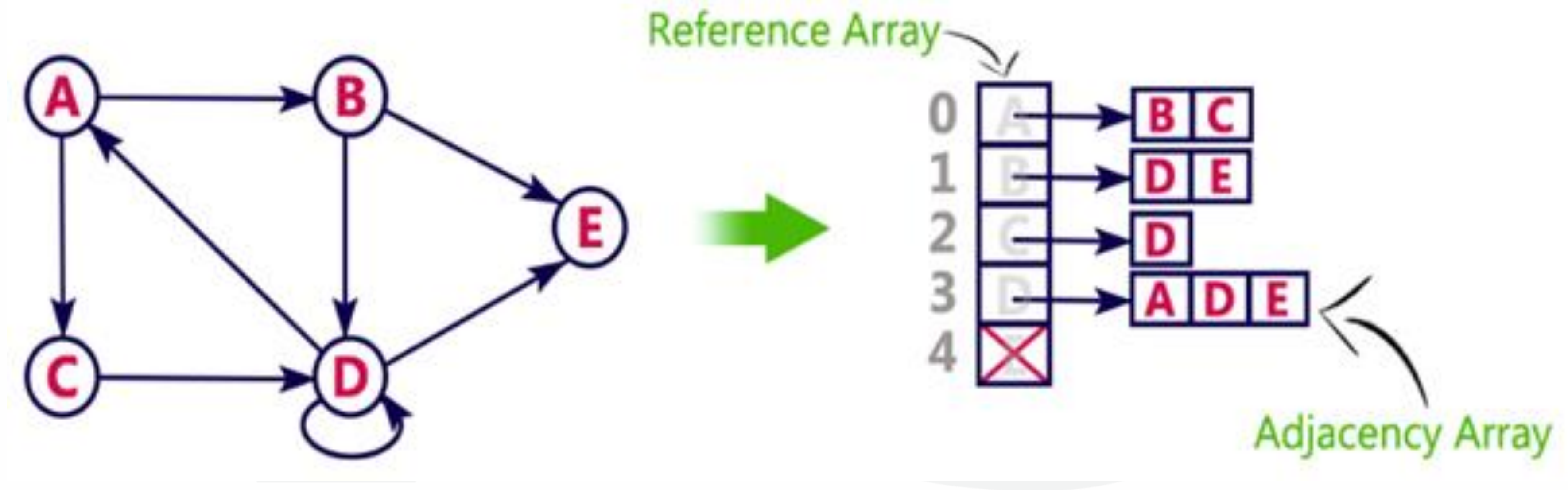
- Adjacency list is a linked representation.
- In this representation, for each vertex in the graph, we maintain the list of its neighbors. It means, every vertex of the graph contains list of its adjacent vertices.
- We have an array of vertices which is indexed by the vertex number and for each vertex v, the corresponding array element points to a singly linked list of neighbors of v.



Directed Graph

Representation of Graph (Using Adjacency List)

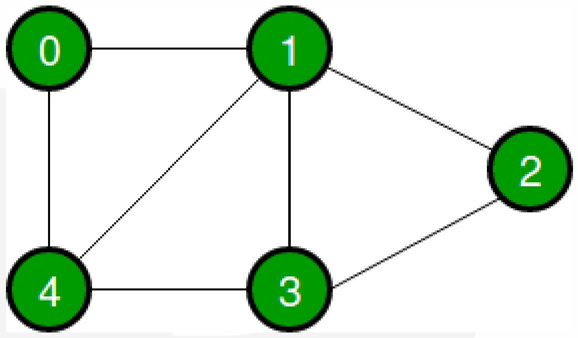
- We can also implement this representation using array as follows:



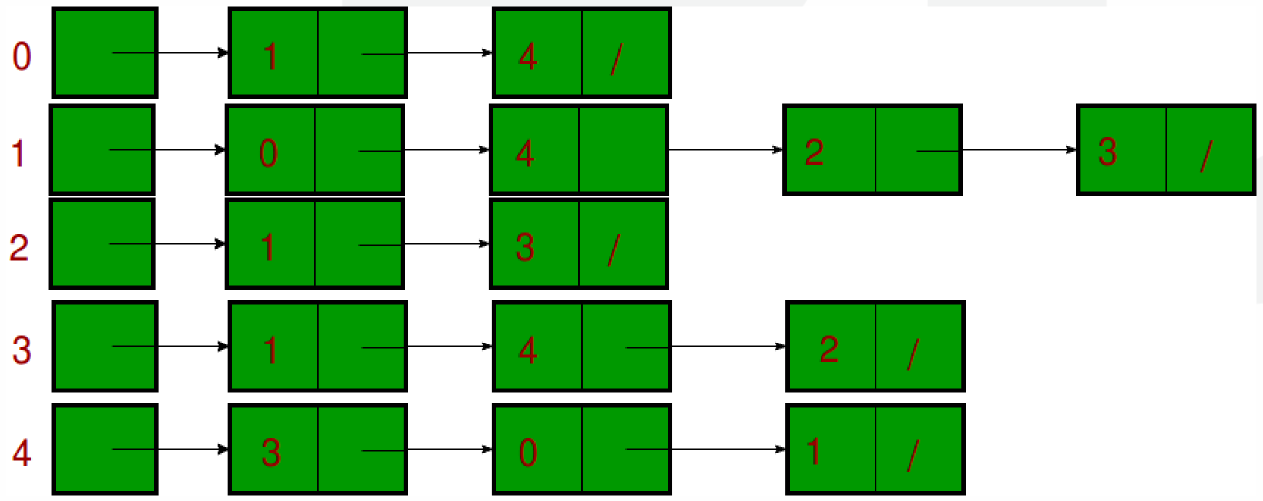
Directed Graph

Representation of Graph (Example)

- Example:



$$
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 \\
2 & 0 & 1 & 0 & 1 & 0 \\
3 & 0 & 1 & 1 & 0 & 1 \\
4 & 1 & 1 & 0 & 1 & 0
\end{array}
$$



Introduction to Graph Traversal

- Graph traversal is the process of visiting each vertex in a graph. Means navigating through a network of nodes, where each node represents a point of interest and edges represent the connections between them.
- Graph traversal algorithms are fundamental in graph theory and computer science.
- They provide systematic ways to explore graphs, ensuring that each vertex is visited in a specific order. This systematic approach helps in various tasks like searching, pathfinding, and analyzing the connectivity of the graph.
- **Traversal Algorithm:**
  - **Breadth-First Search (BFS)**, explore all neighboring vertices before moving to the next level
  - **Depth-First Search (DFS)**, go as far as possible along each branch before backtracking.

Breadth-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.

Breadth-First Search (BFS)

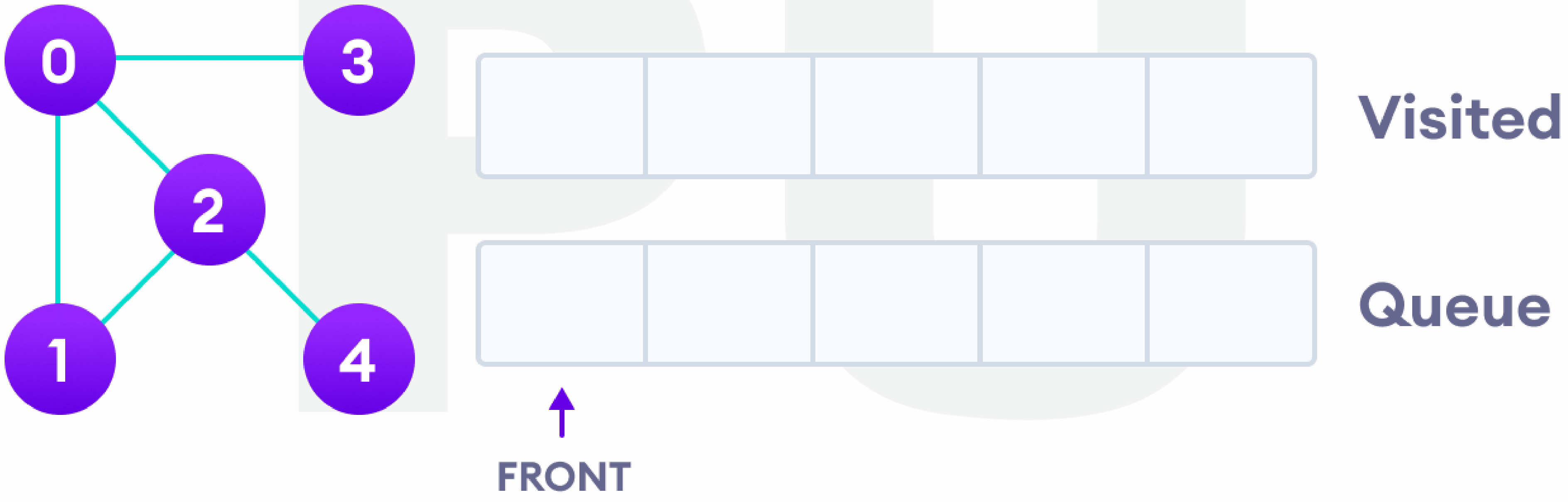
- A standard BFS implementation puts each vertex of the graph into one of two categories:
  1. Visited
  2. Not Visited
- The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.
- The algorithm works as follows:
  1. Start by putting any one of the graph's vertices at the back of a queue.
  2. Take the front item of the queue and add it to the visited list.
  3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
  4. Keep repeating steps 2 and 3 until the queue is empty.

Parul® University

DIGITAL LEARNING CONTENT

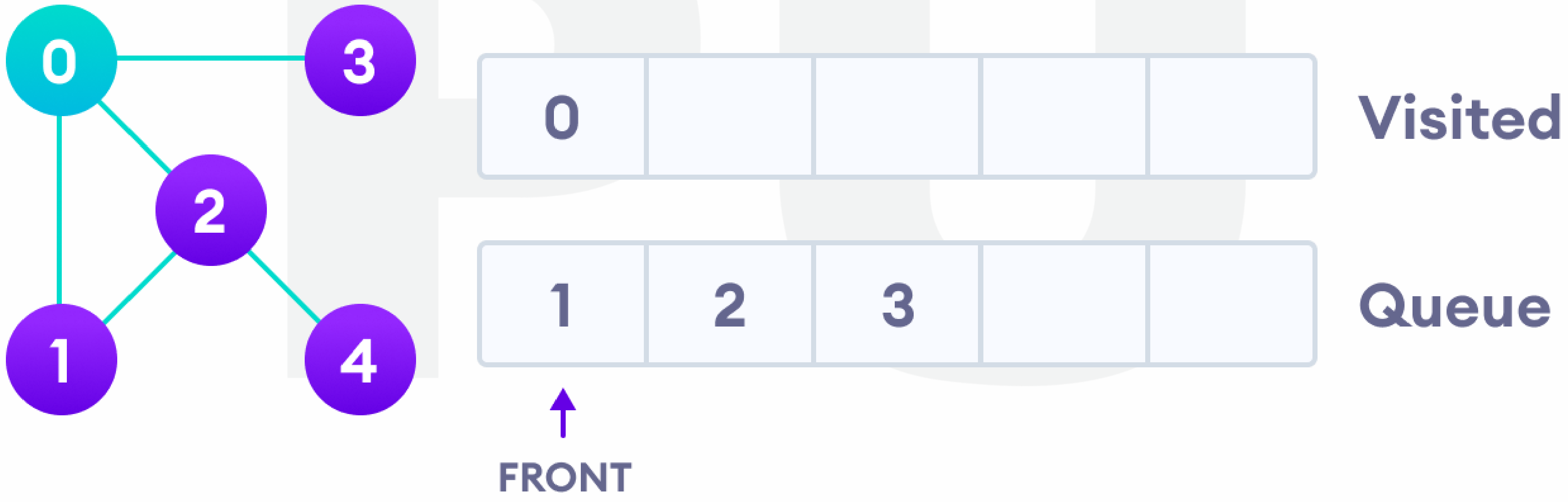
Breadth-First Search (BFS) Example

We use an undirected graph with 5 vertices.



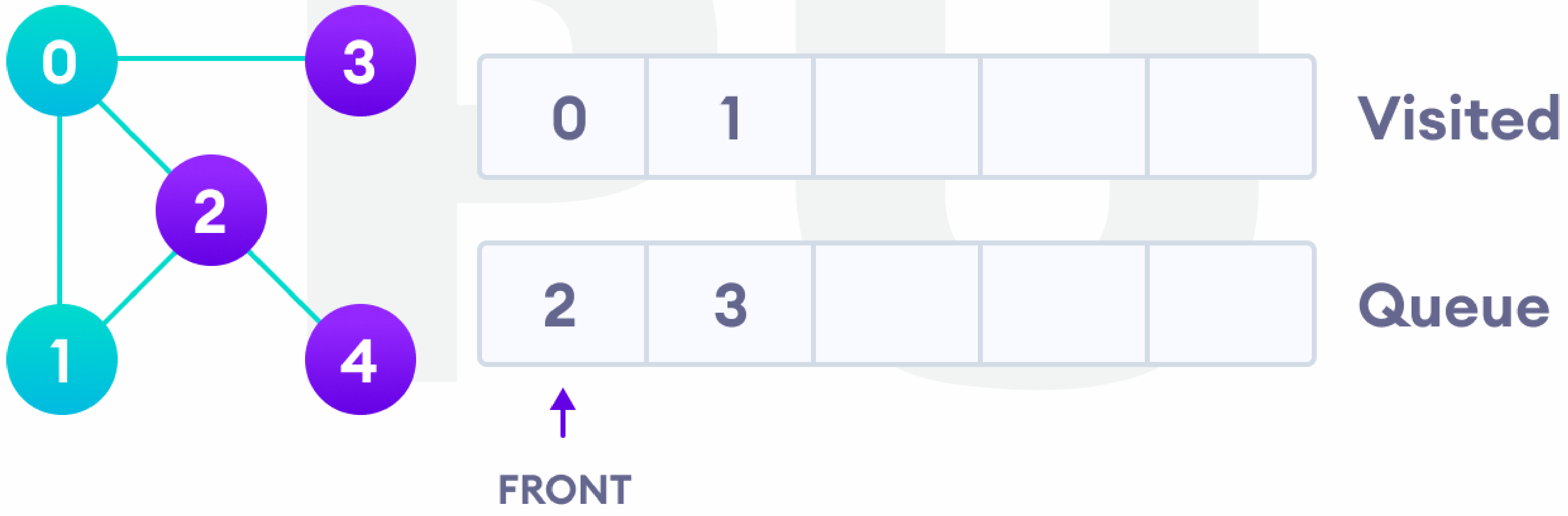
Breadth-First Search (BFS) Example

We start from vertex 0, the BFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the queue.



Breadth-First Search (BFS) Example

Next, we visit the element at the front of queue i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.

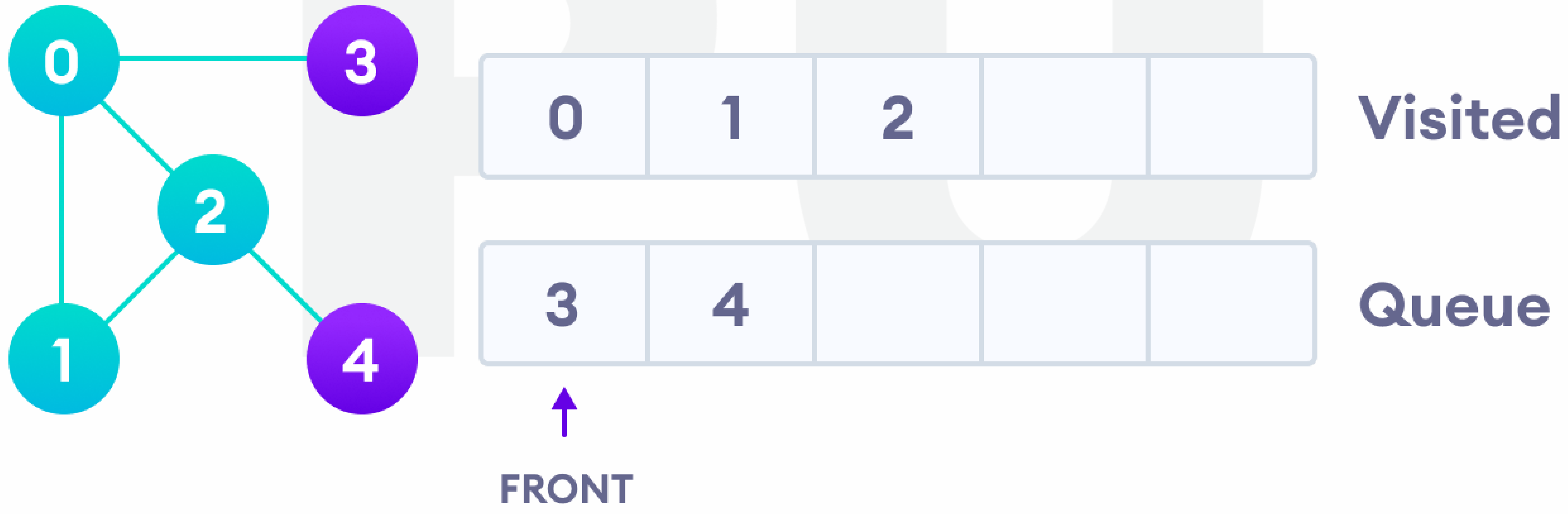


Parul® University

DIGITAL LEARNING CONTENT

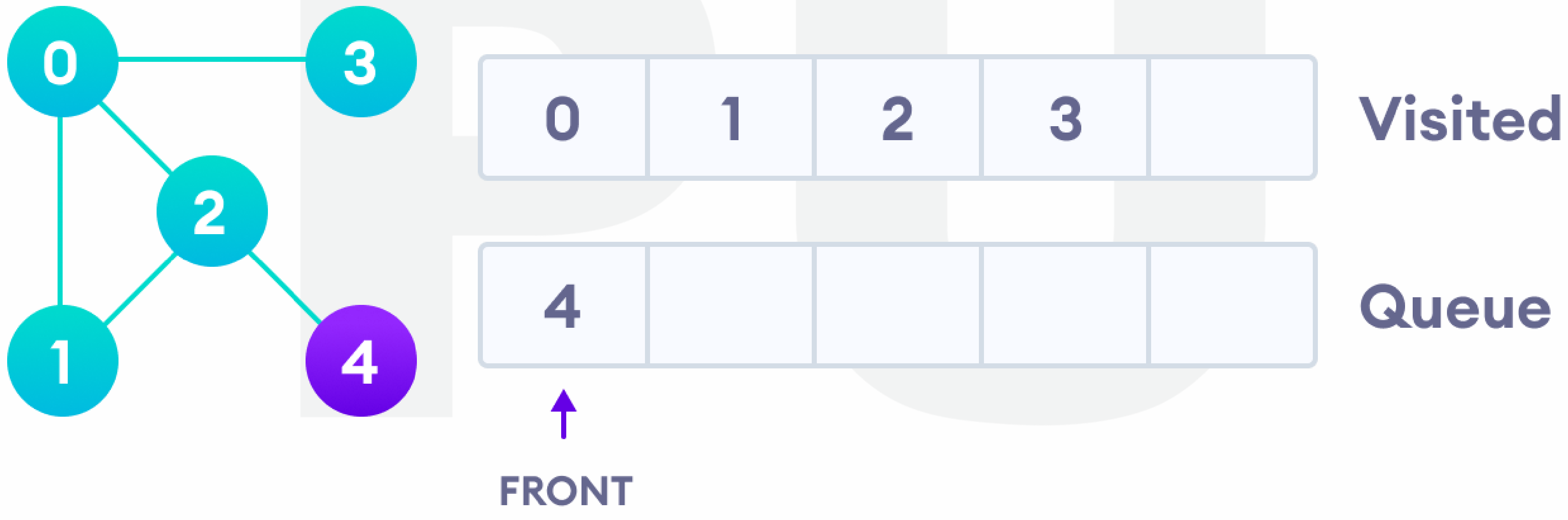
Breadth-First Search (BFS) Example

Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the back of the queue and visit 3, which is at the front of the queue.



Breadth-First Search (BFS) Example

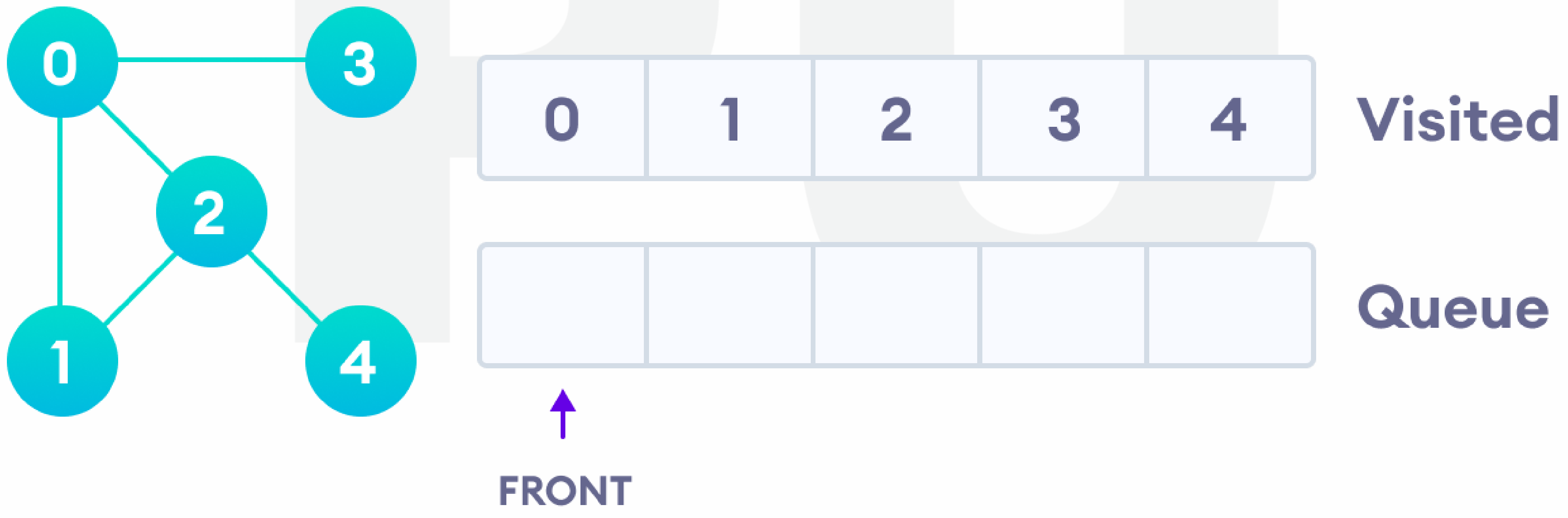
Only 4 remains in the queue since the only adjacent node of 3 i.e. 0 is already visited. We visit it.



Breadth-First Search (BFS) Example

Visit last remaining item in the queue to check if it has unvisited neighbors.

Since the queue is empty, we have completed the Breadth First Traversal of the graph.



Breadth-First Search (BFS) – Applications

- BFS can be used to find the neighboring locations from a given source location.
- In a peer-to-peer network, BFS algorithm can be used as a traversal method to find all the neighboring nodes.
- Most torrent clients, such as BitTorrent, uTorrent, etc. employ this process to find "seeds" and "peers" in the network.
- BFS can be used in web crawlers to create web page indexes. It is one of the main algorithms that can be used to index web pages. It starts traversing from the source page and follows the links associated with the page. Here, every web page is considered as a node in the graph.

Depth-First Search (DFS)

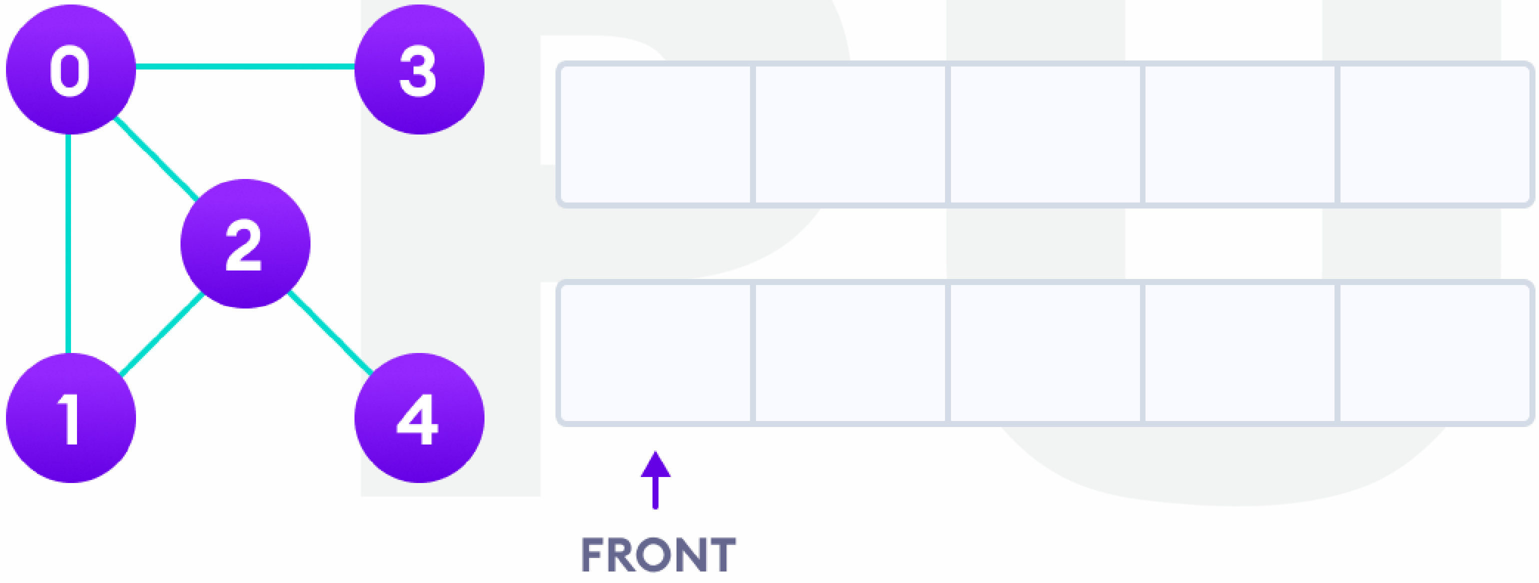
- Depth First Search (DFS) algorithm is a recursive algorithm for searching all the vertices of a graph or tree data structure.
- This algorithm traverses a graph in a deathward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.
- Because of the recursive nature, ***stack data structure*** can be used to implement the DFS algorithm. The process of implementing the DFS is similar to the BFS algorithm.

Depth-First Search (DFS)

- A standard DFS implementation puts each vertex of the graph into one of two categories:
  1. Visited
  2. Not Visited
- The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.
- The algorithm works as follows:
  1. Start by putting any one of the graph's vertices on top of a stack.
  2. Take the top item of the stack and add it to the visited list.
  3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.
  4. Keep repeating steps 2 and 3 until the stack is empty.

Depth-First Search (DFS) Example

We use an undirected graph with 5 vertices.

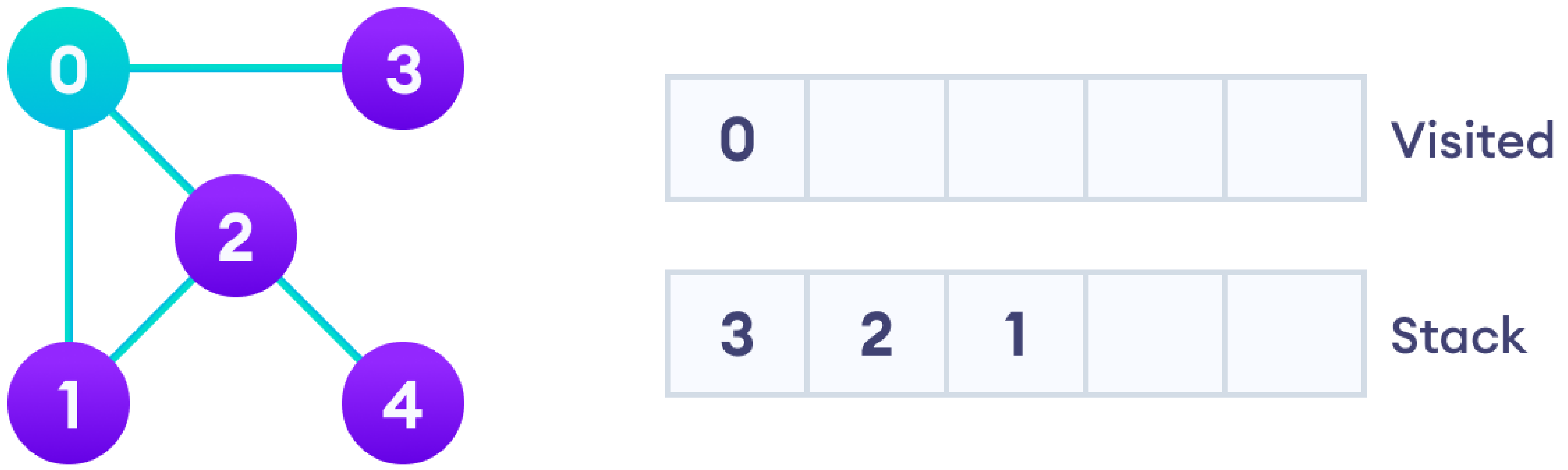


Parul® University

DIGITAL LEARNING CONTENT

Depth-First Search (DFS) Example

We start from vertex 0, the DFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.



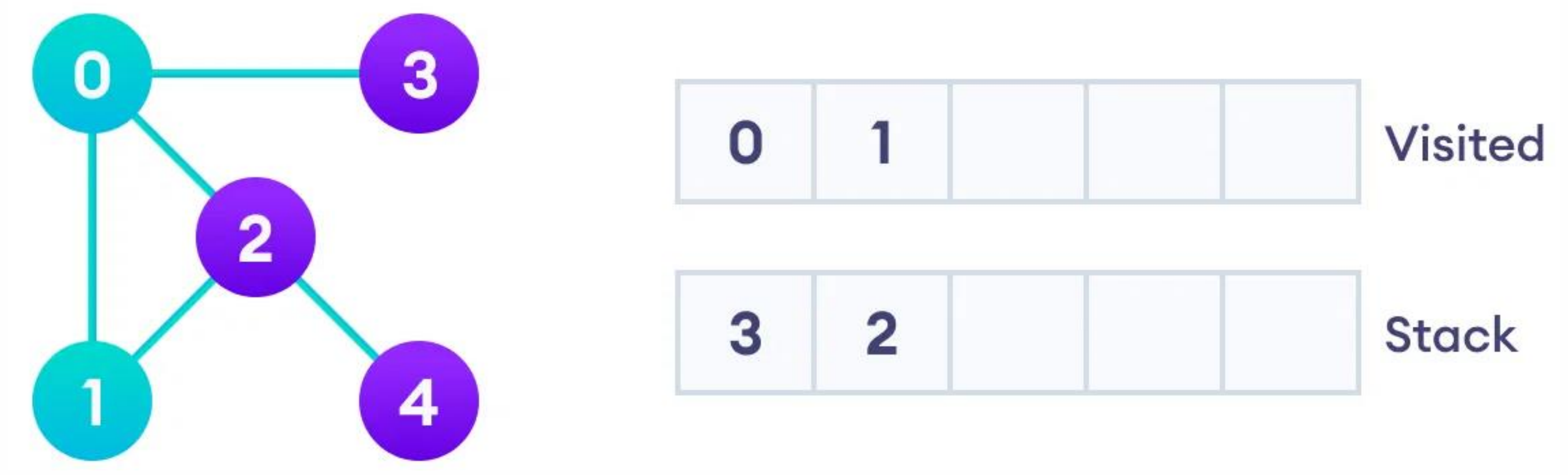
0					Visited
3	2	1			Stack

Parul® University

DIGITAL LEARNING CONTENT

Depth-First Search (DFS) Example

Next, we visit the element at the top of stack i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.

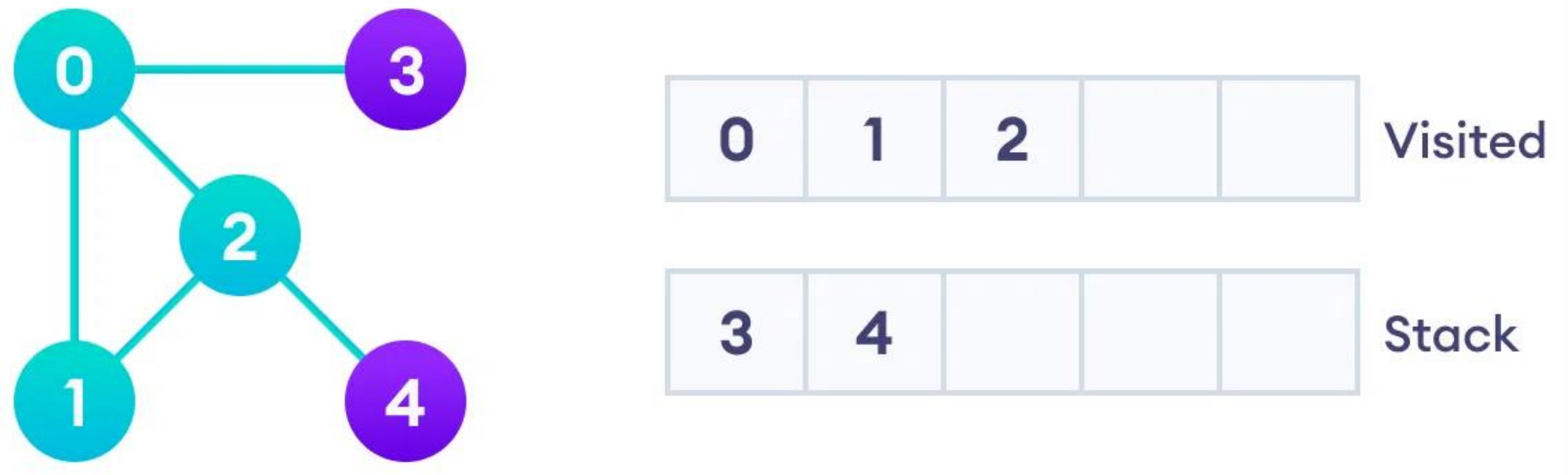


Parul® University

DIGITAL LEARNING CONTENT

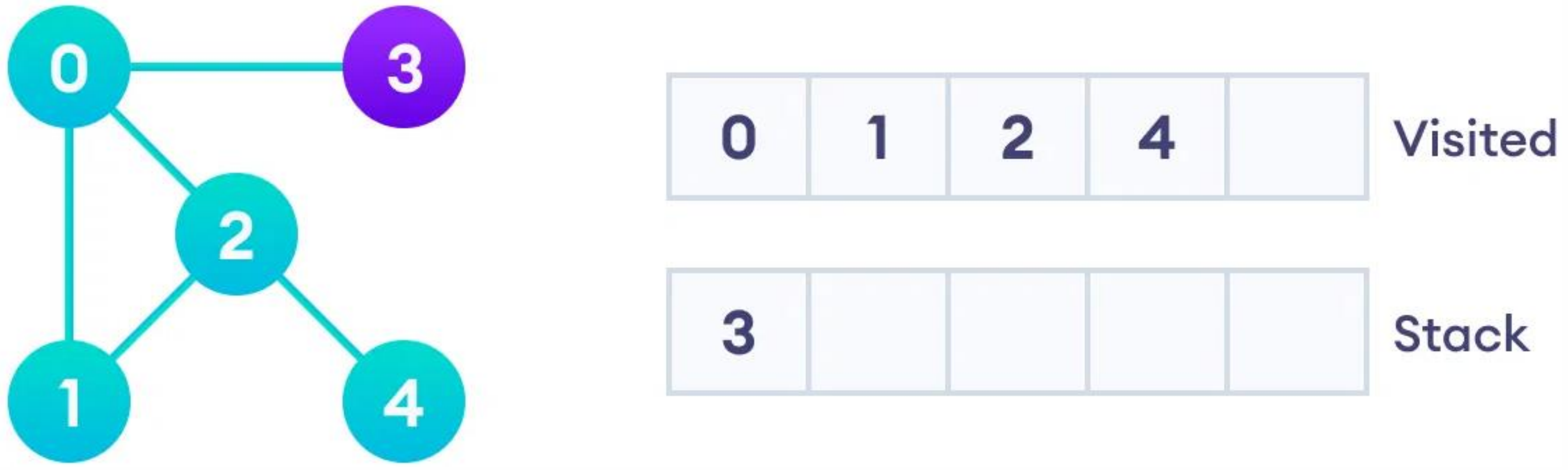
Depth-First Search (DFS) Example

Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.



Depth-First Search (DFS) Example

Only 3 remains in the queue since the only adjacent node of 0 i.e. 0 is already visited. We visit it.



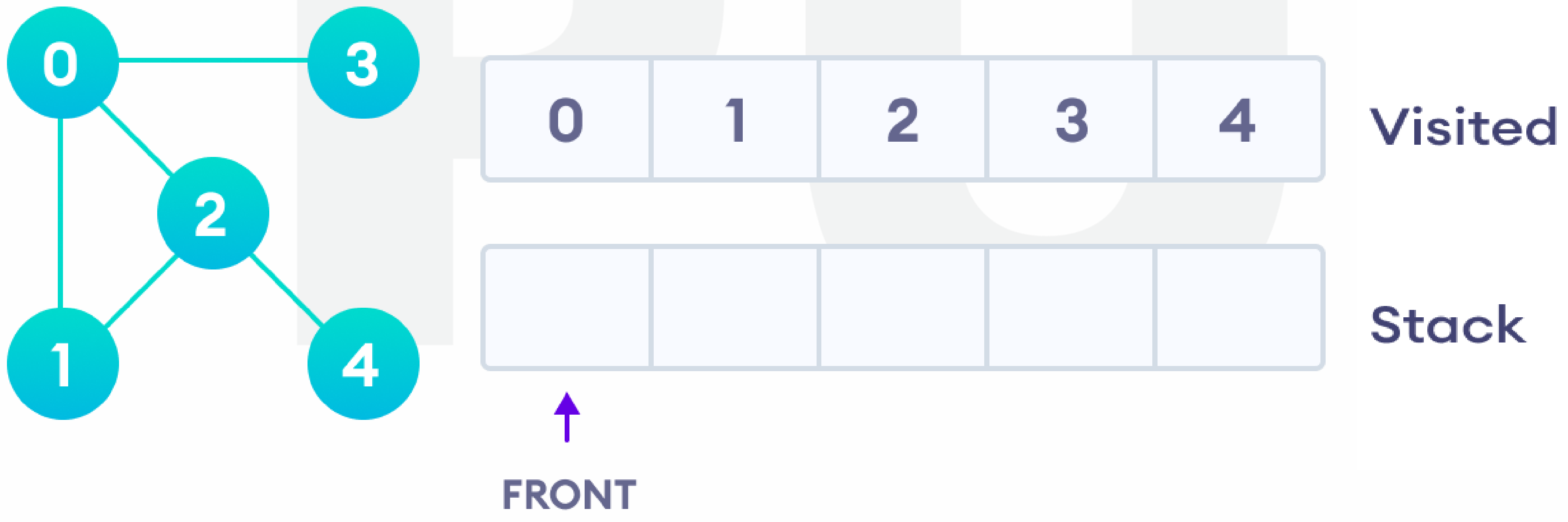
Parul® University

DIGITAL LEARNING CONTENT

Depth-First Search (DFS) Example

Visit last remaining item in the stack to check if it has unvisited neighbors.

Since the stack is empty, we have completed the Depth First Traversal of the graph.

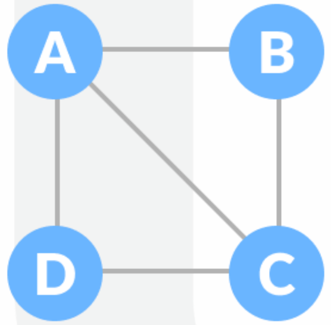


Depth-First Search (DFS) Applications

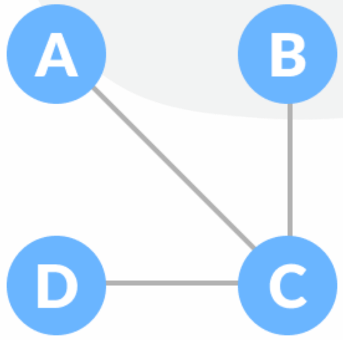
1. For finding the path
2. To test if the graph is bipartite
3. For finding the strongly connected components of a graph
4. For detecting cycles in a graph

Spanning Tree and Minimum Spanning Tree

- Before we learn about spanning trees, we need to understand two graphs: undirected graphs and connected graphs
- An **undirected graph** is a graph in which the edges do not point in any direction (ie. the edges are bidirectional).



- A **connected graph** is a graph in which there is always a path from a vertex to any other vertex.



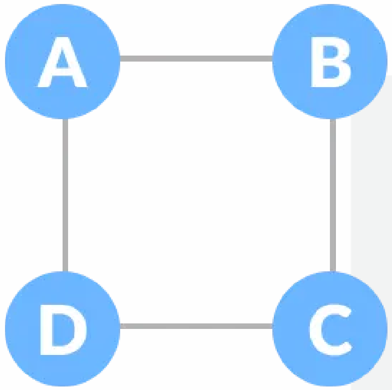
Spanning Tree and Minimum Spanning Tree

- A spanning tree is a sub-graph of an undirected connected graph, which includes all the vertices of the graph with a minimum possible number of edges. If a vertex is missed, then it is not a spanning tree.
- The total number of spanning trees with n vertices that can be created from a complete graph is equal to $n^{(n-2)}$.
- If we have n = 4, the maximum number of possible spanning trees is equal to $4^{4-2} = 16$. Thus, 16 spanning trees can be formed from a complete graph with 4 vertices.

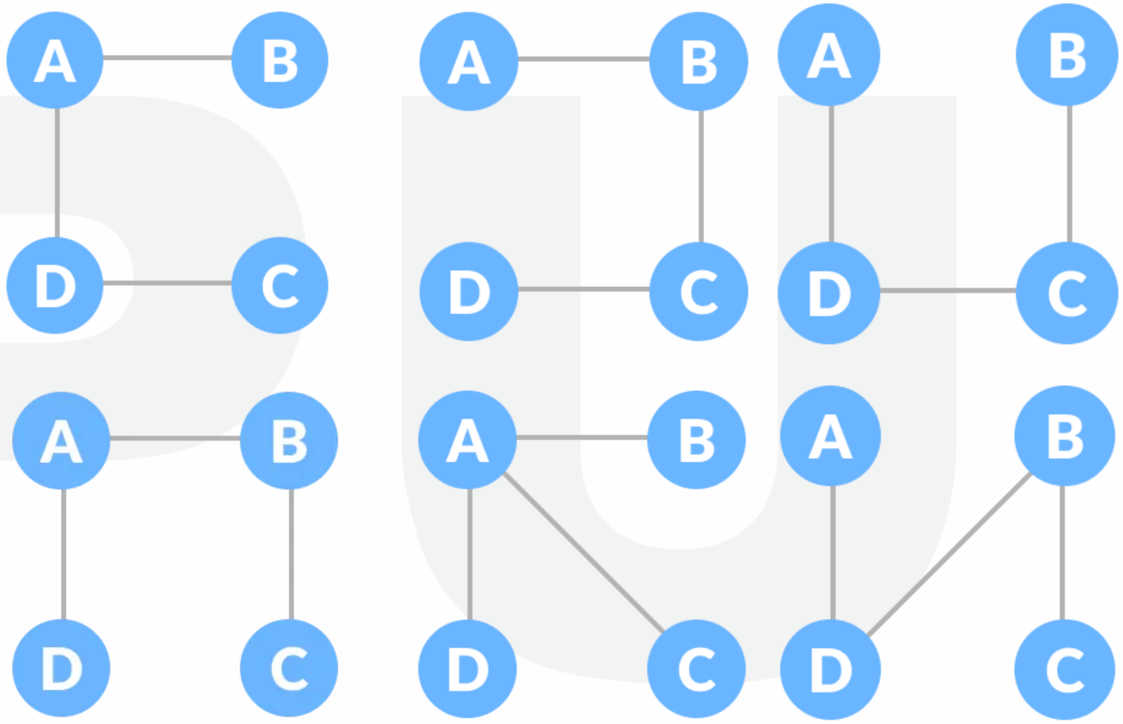
Parul® University

DIGITAL LEARNING CONTENT

Example of Spanning Tree



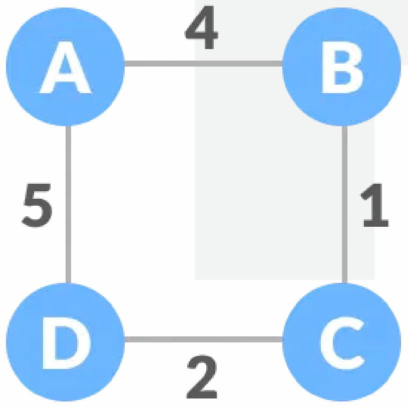
Normal Graph



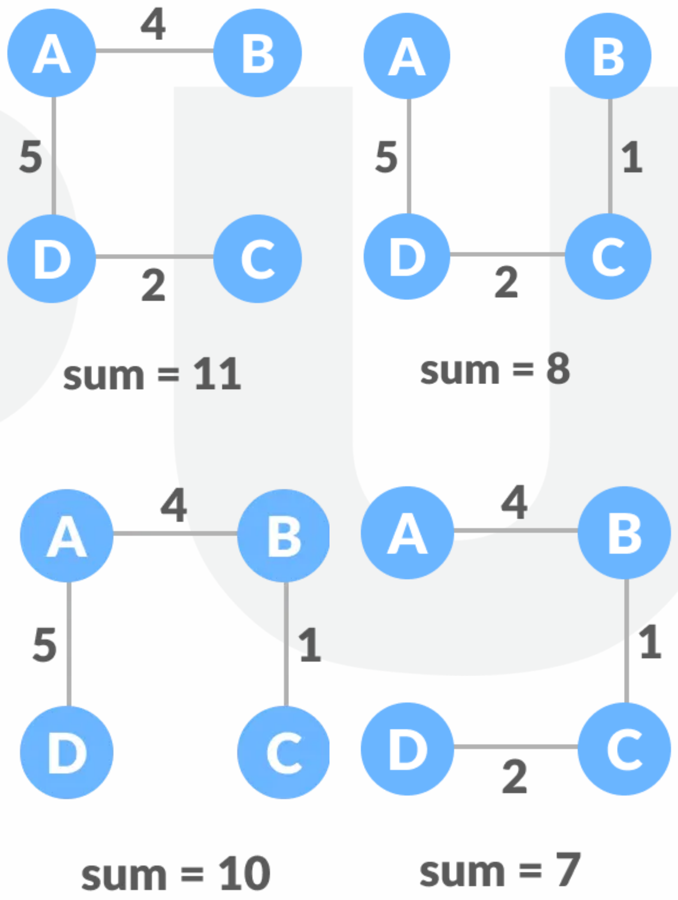
Spanning Tree

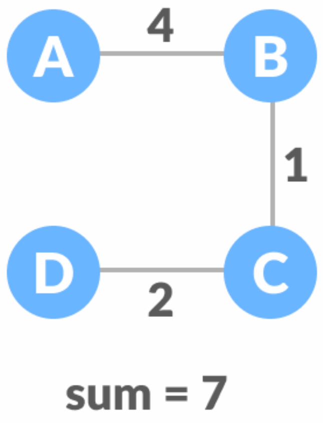
Minimum Spanning Tree

- A minimum spanning tree is a spanning tree in which the sum of the weight of the edges is as minimum as possible.



Weighted graph





Above is the minimum Spanning Tree

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in