

Unit 6: Graph

Prepared By:

Tanvi Patel

Asst. Prof. (CSE)

Contents

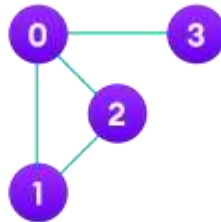
- Graph Terminologies
- Adjacency Matrices
- List Representations of Graphs
- Elementary Graph Operations: Depth First Search & Breadth first Search,
- Spanning Trees: Shortest path,
- Minimal spanning tree using graphs, networks.

Introduction

- A graph data structure is a collection of nodes that have data and are connected to other nodes.
- Let's try to understand this through an example. On facebook, everything is a node. That includes User, Photo, Album, Event, Group, Page, Comment, Story, Video, Link, Note...anything that has data is a node.
- Every relationship is an edge from one node to another. Whether you post a photo, join a group, like a page, etc., a new edge is created for that relationship.

Introduction

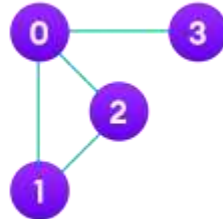
- All of facebook is then a collection of these nodes and edges. This is because facebook uses a graph data structure to store its data.
- More precisely, a graph is a data structure (V, E) that consists of
- A collection of vertices V
- A collection of edges E , represented as ordered pairs of vertices (u,v)
- In the graph,
- $V = \{0, 1, 2, 3\}$
- $E = \{(0,1), (0,2), (0,3), (1,2)\}$
- $G = \{V, E\}$



Graph Terminologies

- ◉ **Adjacency:** A vertex is said to be adjacent to another vertex if there is an edge connecting them. Vertices 2 and 3 are not adjacent because there is no edge between them.
- ◉ **Path:** A sequence of edges that allows you to go from vertex A to vertex B is called a path. 0-1, 1-2 and 0-2 are paths from vertex 0 to vertex 2.
- ◉ **Directed Graph:** A graph in which an edge (u,v) doesn't necessarily mean that there is an edge (v,u) as well. The edges in such a graph are represented by arrows to show the direction of the edge.

◉



Depth First Search (DFS)

- Depth first Search or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure. Traversal means visiting all the nodes of a graph.

Depth First Search Algorithm

- A standard DFS implementation puts each vertex of the graph into one of two categories:

Visited

Not Visited

- The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.
- The DFS algorithm works as follows:

Start by putting any one of the graph's vertices on top of a stack.

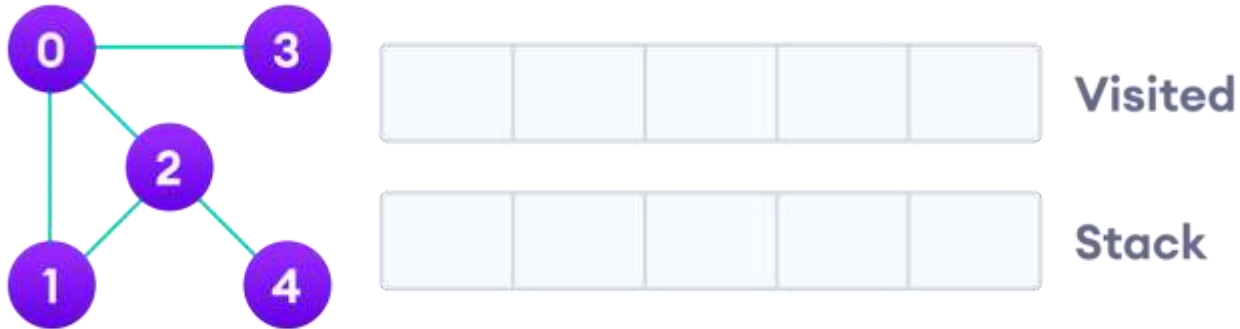
Take the top item of the stack and add it to the visited list.

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.

Keep repeating steps 2 and 3 until the stack is empty.

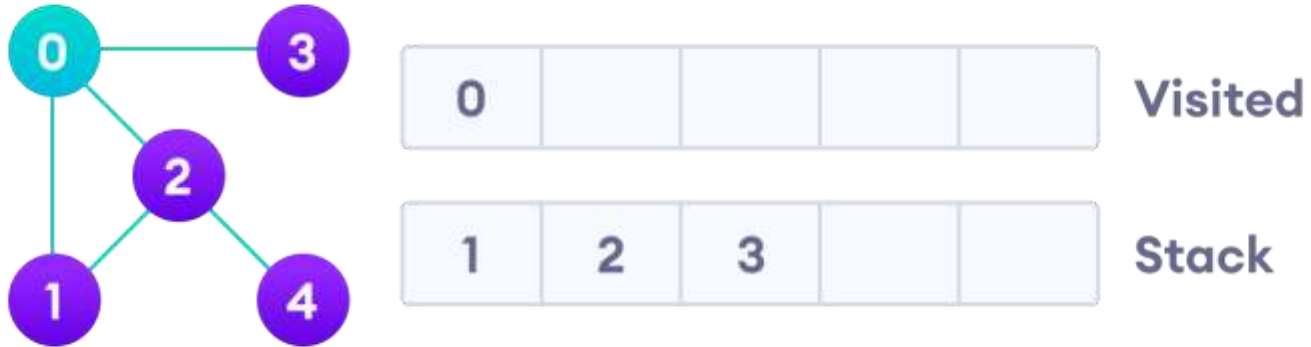
Depth First Search (DFS)

- **Depth First Search Example:** Let's see how the Depth First Search algorithm works with an example. We use an undirected graph with 5 vertices.



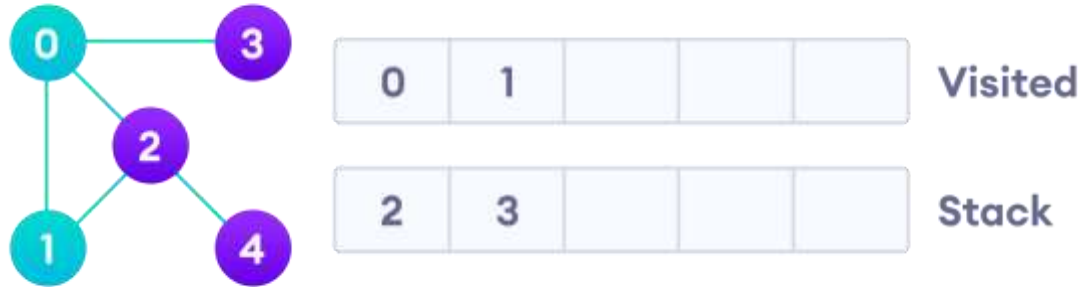
Depth First Search (DFS)

- 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.



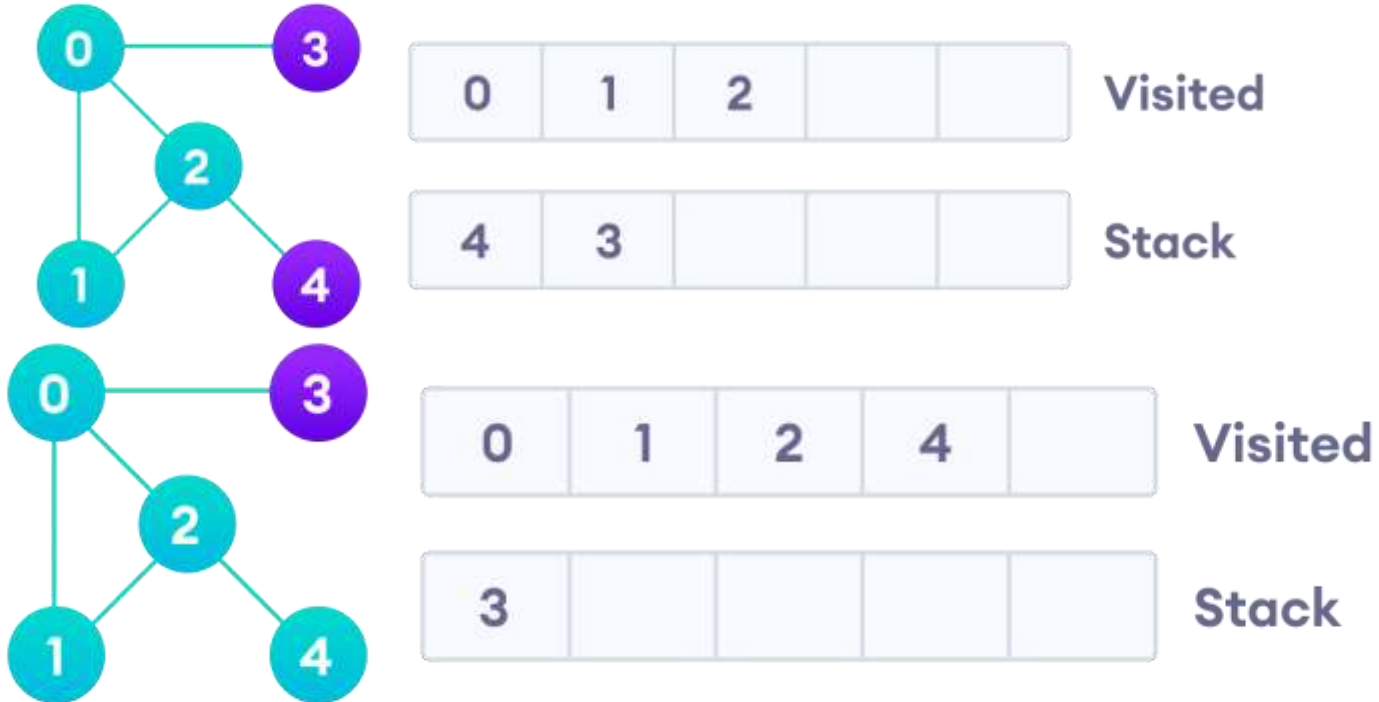
Depth First Search (DFS)

- 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.



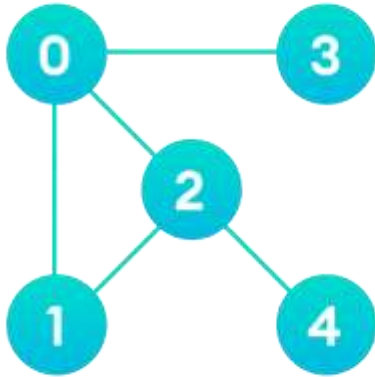
Depth First Search (DFS)

- 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)

- After we visit the last element 3, it doesn't have any unvisited adjacent nodes, so we have completed the Depth First Traversal of the graph.



Visited



Stack

Depth First Search (DFS)

Complexity of Depth First Search

- The time complexity of the DFS algorithm is represented in the form of $O(V + E)$, where V is the number of nodes and E is the number of edges.

Breadth first search

- Breadth First Traversal or Breadth First Search is a recursive algorithm for searching all the vertices of a graph or tree data structure.

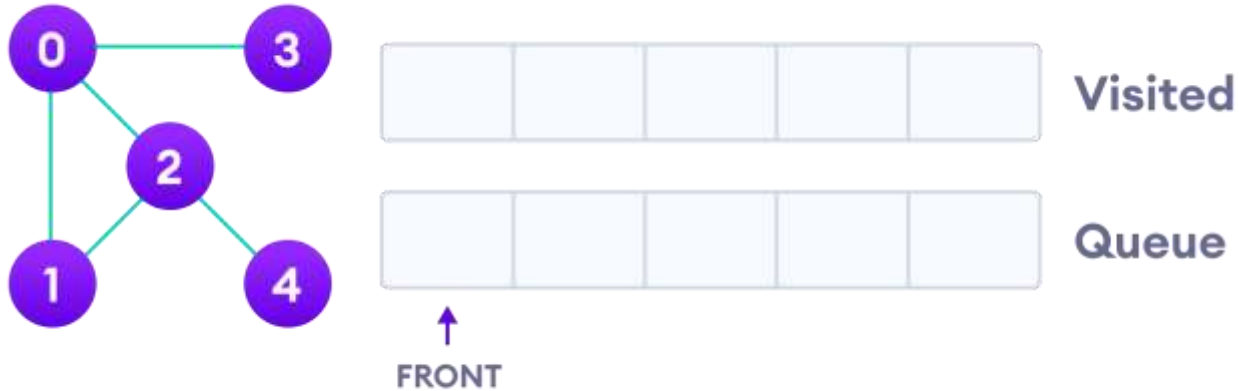
BFS algorithm

- A standard BFS implementation puts each vertex of the graph into one of two categories:
Visited
Not Visited
- The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.
- The algorithm works as follows:
Start by putting any one of the graph's vertices at the back of a queue.
Take the front item of the queue and add it to the visited list.
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.
Keep repeating steps 2 and 3 until the queue is empty.
- The graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the BFS algorithm on every node

Breadth first search

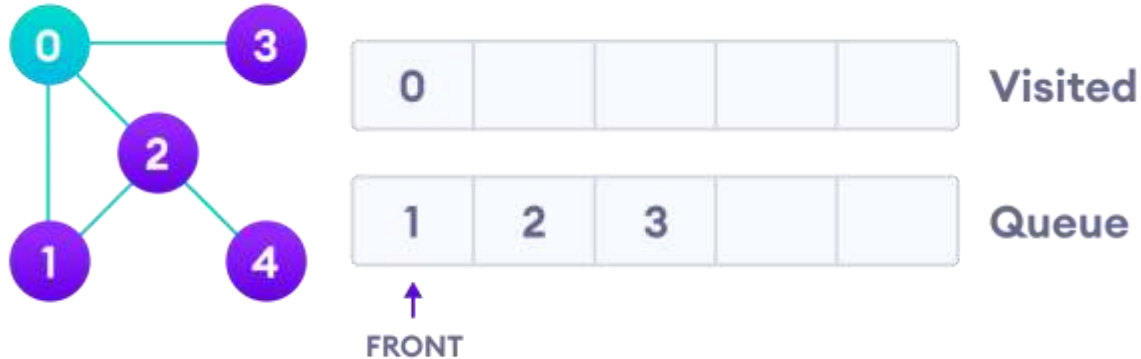
BFS example

- Let's see how the Breadth First Search algorithm works with an example. We use an undirected graph with 5 vertices.



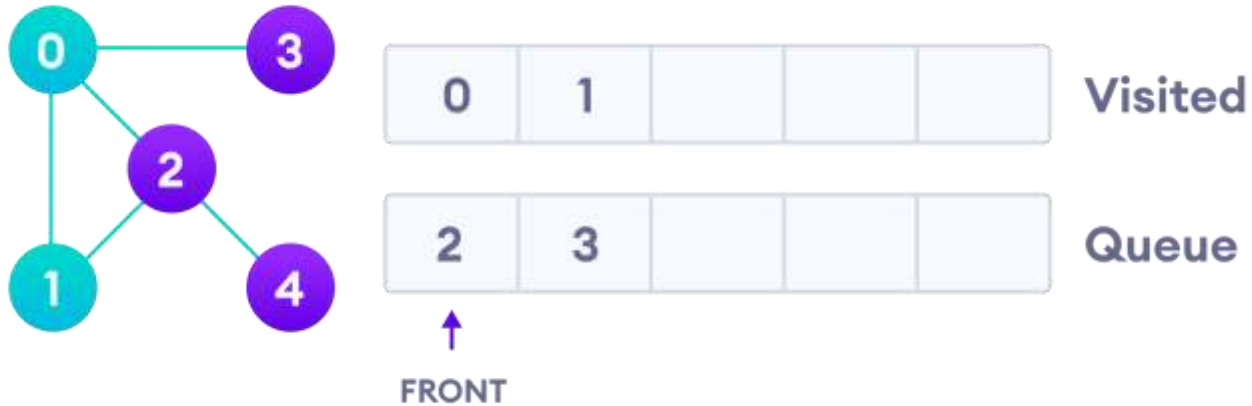
Breadth first search

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.



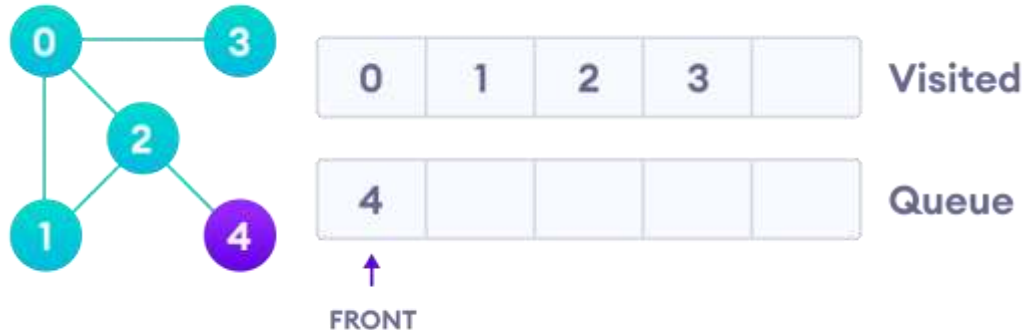
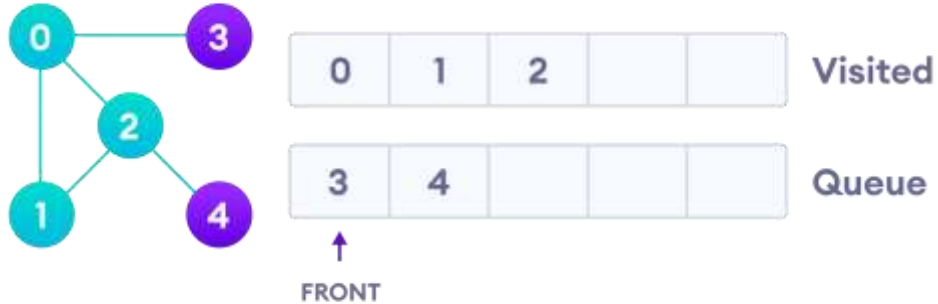
Breadth first search

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.



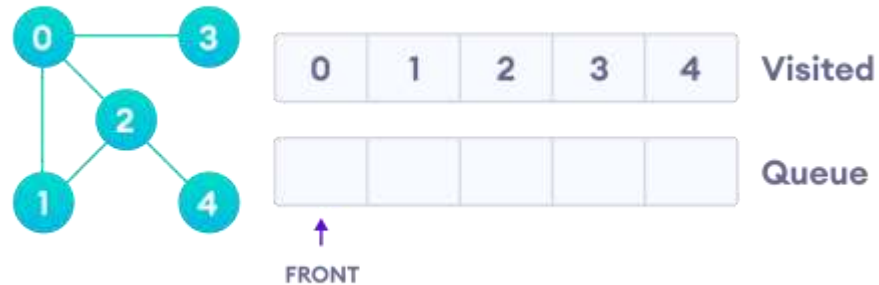
Breadth first search

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

Only 4 remains in the queue since the only adjacent node of 3 i.e. 0 is already visited. We visit it. Since the queue is empty, we have completed the Breadth First Traversal of the graph.



Breadth first search

BFS Algorithm Complexity

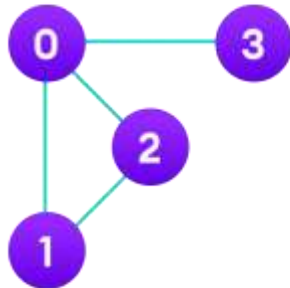
- ◉ The time complexity of the BFS algorithm is represented in the form of $O(V + E)$, where V is the number of nodes and E is the number of edges.

Graph Representation

Graphs are commonly represented in two ways:

1. Adjacency Matrix

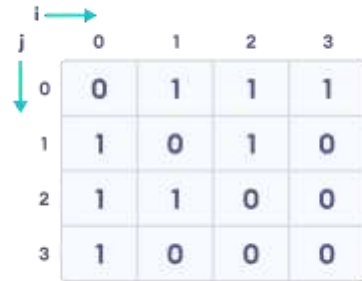
- An adjacency matrix is a 2D array of $V \times V$ vertices. Each row and column represent a vertex.
- An adjacency matrix is a way of representing a graph as a matrix of booleans (0's and 1's). A finite graph can be represented in the form of a square matrix on a computer, where the boolean value of the matrix indicates if there is a direct path between two vertices.
- If the value of any element $a[i][j]$ is 1, it represents that there is an edge connecting vertex i and vertex j .
- The adjacency matrix for the graph we created above is



	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

Graph Representation

- Each cell in the below table/matrix is represented as A_{ij} , where i and j are vertices. The value of A_{ij} is either 1 or 0 depending on whether there is an edge from vertex i to vertex j .



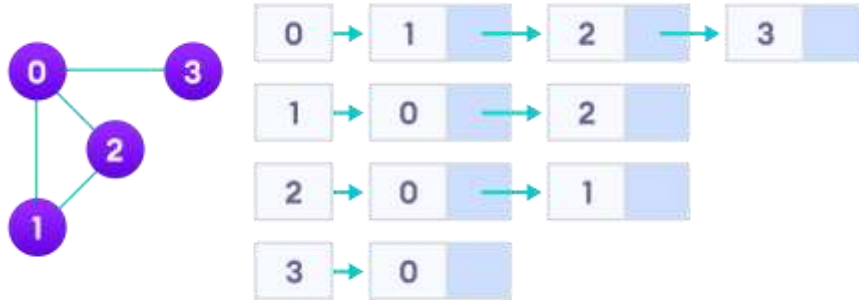
j	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

- If there is a path from i to j , then the value of A_{ij} is 1 otherwise its 0. For instance, there is a path from vertex 1 to vertex 2, so A_{12} is 1 and there is no path from vertex 1 to 3, so A_{13} is 0.
- In case of undirected graphs, the matrix is symmetric about the diagonal because of every edge (i,j) , there is also an edge (j,i) .

Graph Representation

2. Adjacency List

- An adjacency list represents a graph as an array of linked lists.
- The index of the array represents a vertex and each element in its linked list represents the other vertices that form an edge with the vertex.
- The adjacency list for the graph we made in the first example is as follows:



Graph Representation

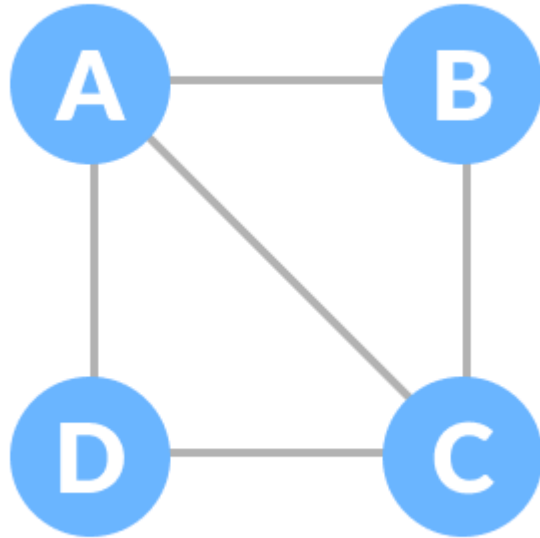
Here, **0, 1, 2, 3** are the vertices and each of them forms a linked list with all of its adjacent vertices. For instance, vertex 1 has two adjacent vertices 0 and 2. Therefore, 1 is linked with 0 and 2 in the figure above.

A simple dictionary of vertices and its edges is a sufficient representation of a graph.

```
graph = { 'A': set(['B', 'C']),  
          'B': set(['A', 'D', 'E']),  
          'C': set(['A', 'F']),  
          'D': set(['B']),  
          'E': set(['B', 'F']),  
          'F': set(['C', 'E'])  
}
```

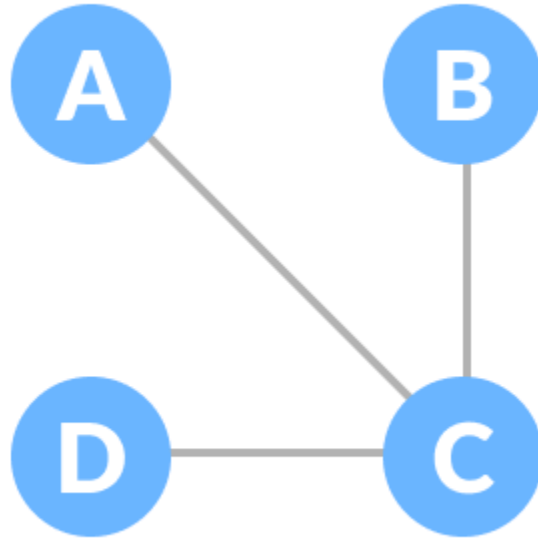
Spanning Tree and Minimum Spanning Tree

- 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).



Spanning Tree and Minimum Spanning Tree

- 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

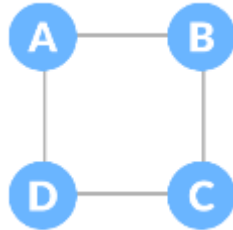
Spanning tree

- 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 edges may or may not have weights assigned to them.
- 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.

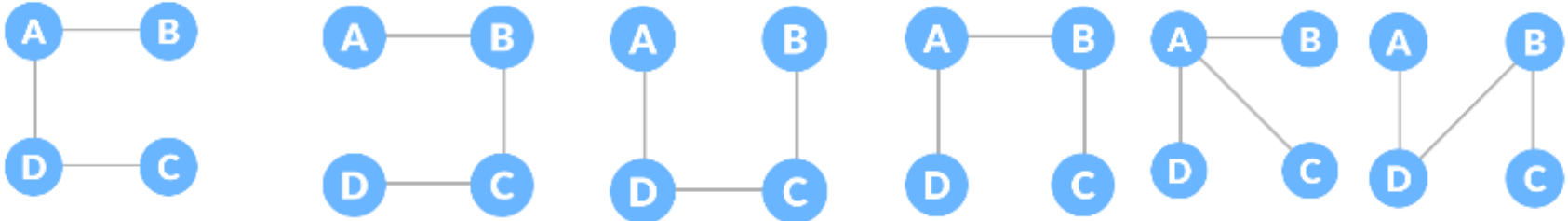
Spanning Tree and Minimum Spanning Tree

Example of a Spanning Tree

- Let the original graph be:



- Some of the possible spanning trees that can be created from the above graph are:



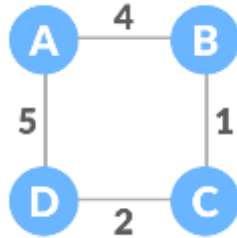
Spanning Tree and Minimum Spanning Tree

Minimum Spanning Tree

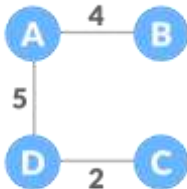
- A minimum spanning tree is a spanning tree in which the sum of the weight of the edges is as minimum as possible.

Example of a Spanning Tree

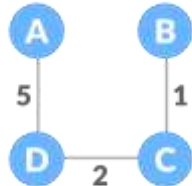
- The initial graph is:



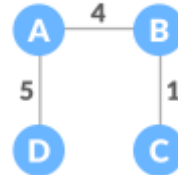
- The possible spanning trees from the above graph are:



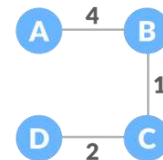
sum = 11



sum = 8



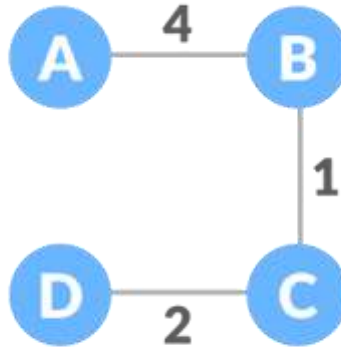
sum = 10



sum = 7

Spanning Tree and Minimum Spanning Tree

The minimum spanning tree from the above spanning trees is:



sum = 7

Thank You