Powerpuff Girls

October 4, 2022

Graph Based DSA

Today's Topic

1. Graph Data Structure
2. Spanning Tree and Minimum Spanning Tree
3. Strongly Connected Components
4. Adjacency Matrix
5. Adjacency List
6. Depth First Search (DFS)
7. Breadth First Search
8. Bellman Ford's Algorithm

Graph Data Structure

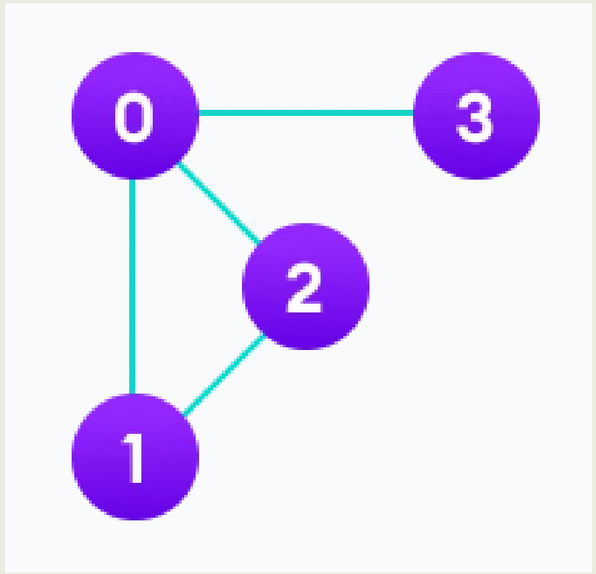
Are you ready?

Graph Data Structure

A graph data structure is a collection of nodes that have data and are connected to other nodes.

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)



Graph Terminology

- **Adjacency:** A vertex is said to be adjacent to another vertex if there is an edge connecting them.

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

- **Path:** A sequence of edges that allows you to go from vertex A to vertex B is called a path.

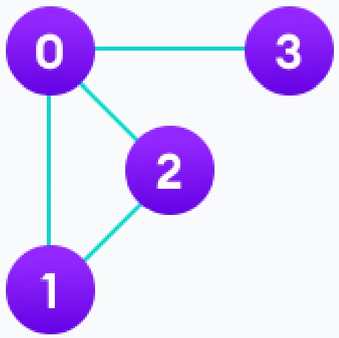
- **Directed Graph:** A graph in which an edge (u,v) doesn't necessarily mean that there is an edge (v, u) as well. The edges in such a graph are represented by arrows to show the direction of the edge.

Graph Representation

1. Adjacency Matrix

An adjacency matrix is a 2D array of V x V vertices. Each row and column represent a vertex.

If the value of any element a[i][j] is 1, it represents that there is an edge connecting vertex i and vertex j.



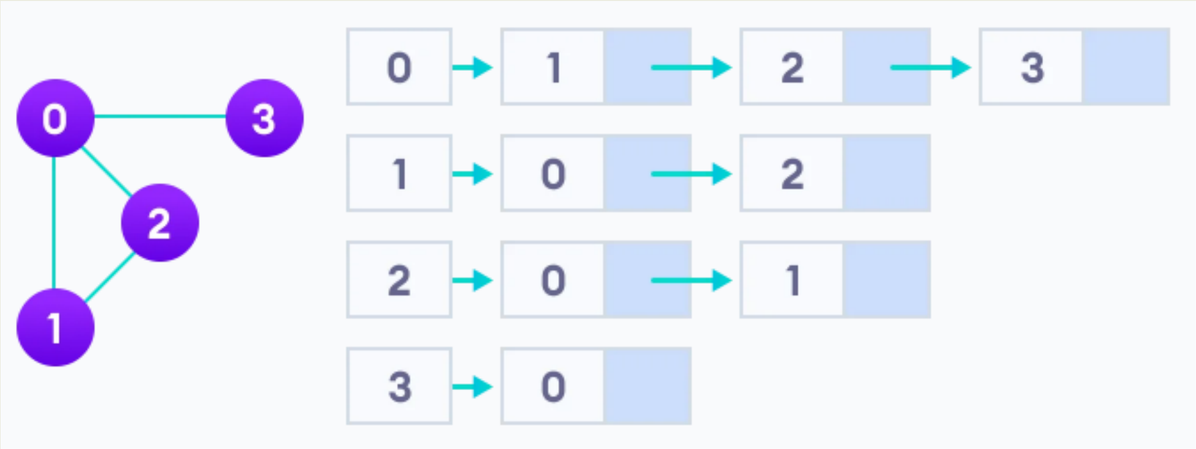
	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

Edge lookup (checking if an edge exists between vertex A and vertex B) is extremely fast in adjacency matrix representation but we have to reserve space for every possible link between all vertices(V x V), so it requires more space.

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.



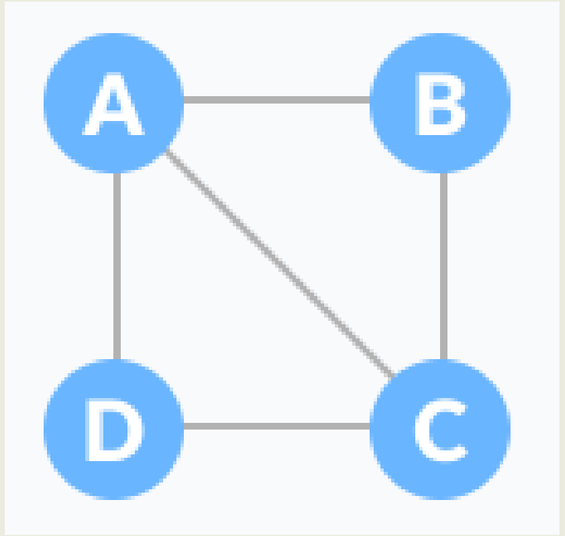
Graph Operations

The most common graph operations are:

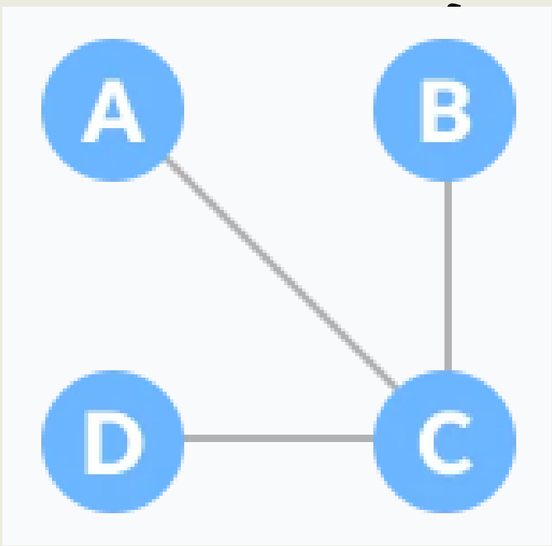
- Check if the element is present in the graph
- Graph Traversal
- Add elements(vertex, edges) to graph
- Finding the path from one vertex to another

Spanning Tree and Minimum Spanning Tree

An **undirected graph** is a graph in which the edges do not point in any direction (ie. the edges are bidirectional).



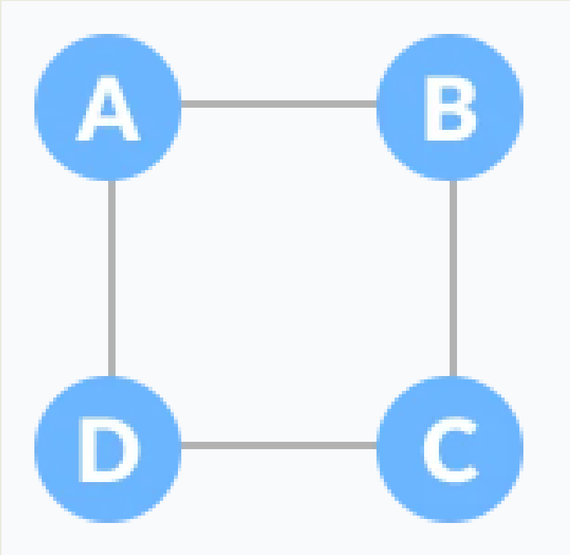
A **connected graph** is a graph in which there is always a path from a vertex to any other vertex.



Spanning Tree

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.

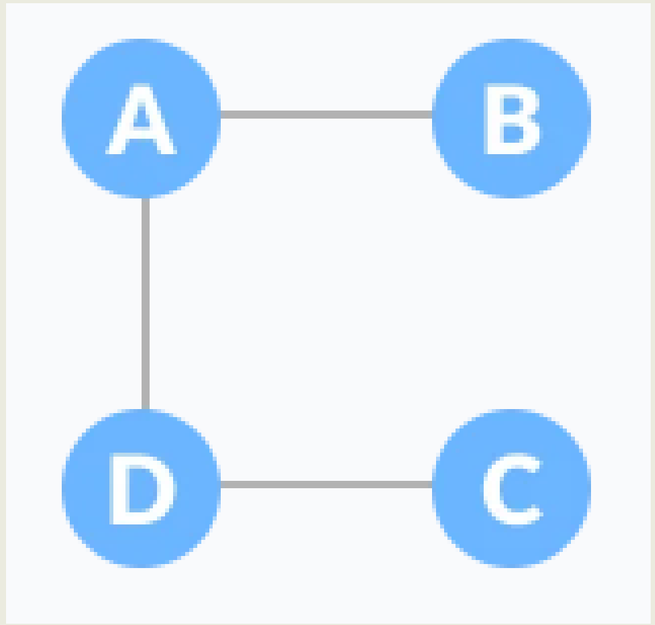
The edges may or may not have weights assigned to them.

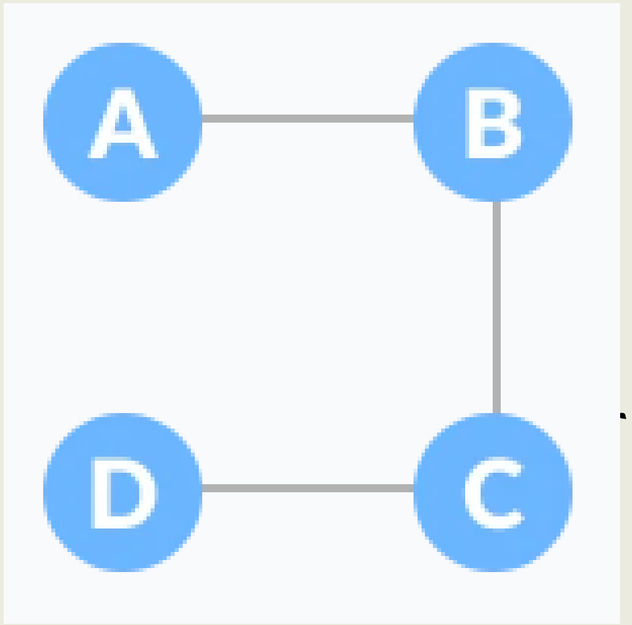


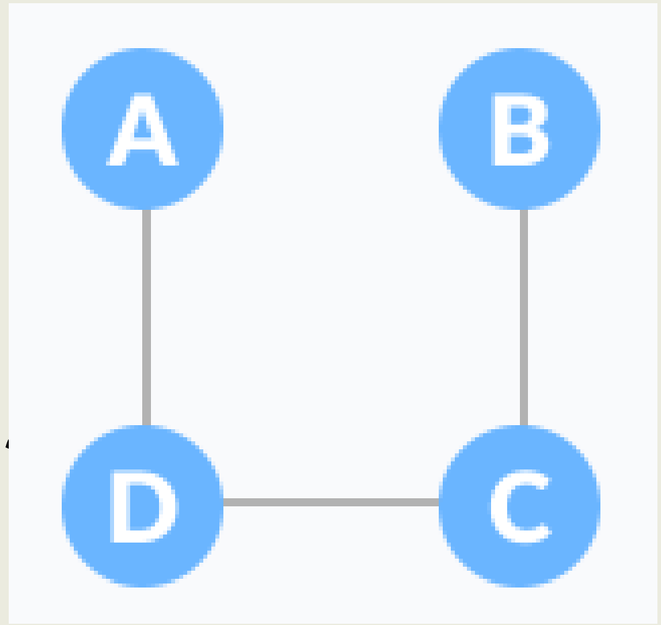
Example of a Spanning Tree

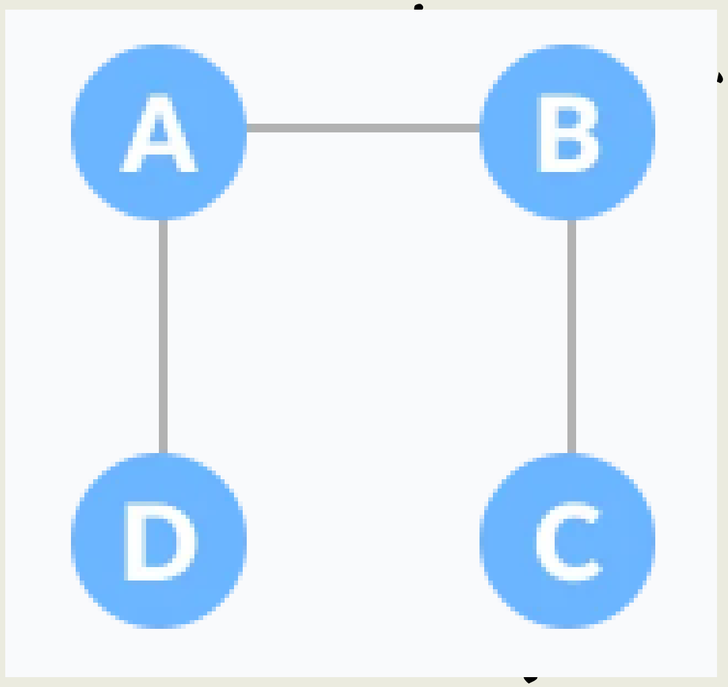
Let the original graph be:

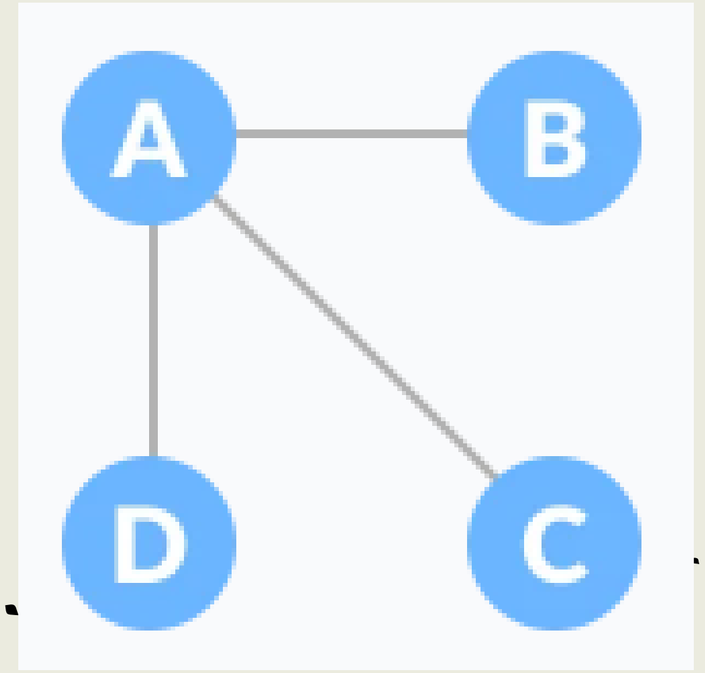
Some of the possible spanning trees that can be created from the above graph are:

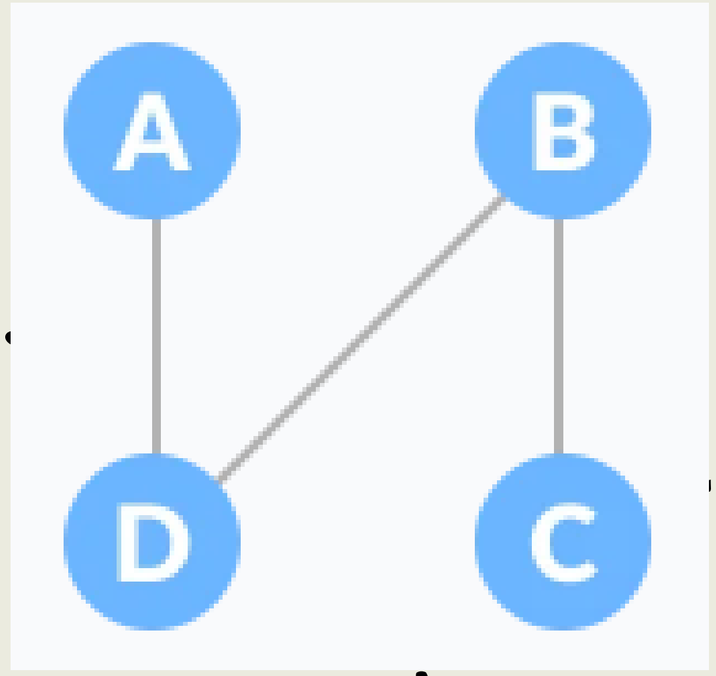






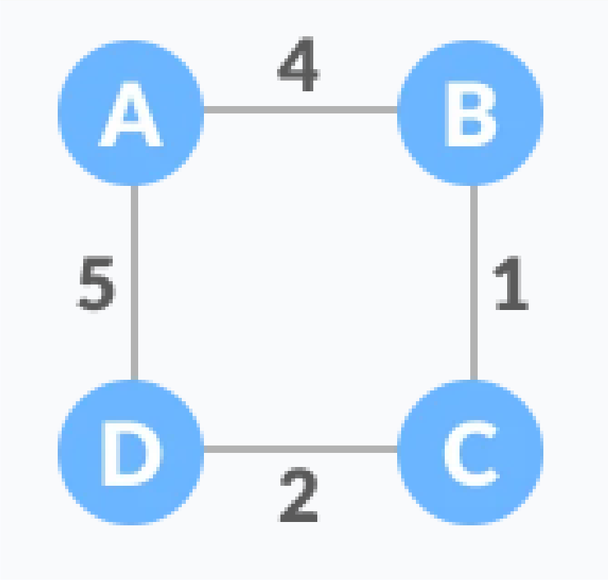

A
B
D
C


A
B
D
C


A
B
D
C

Minimum Spanning Tree

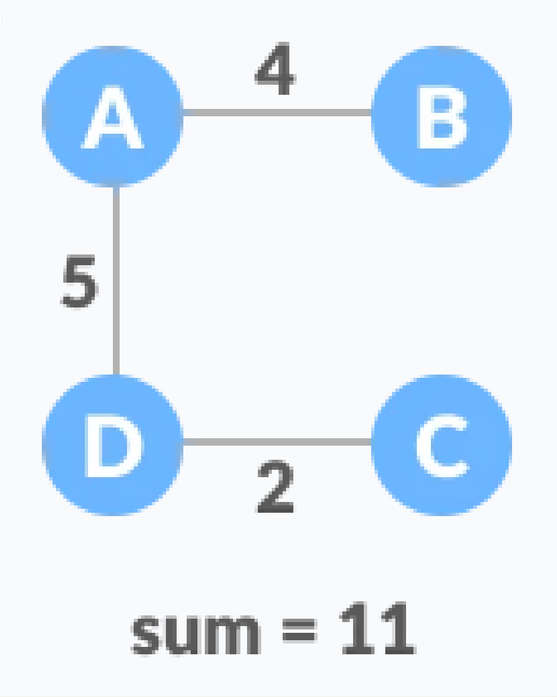
A minimum spanning tree is a spanning tree in which the sum of the weight of the edges is as minimum as possible.

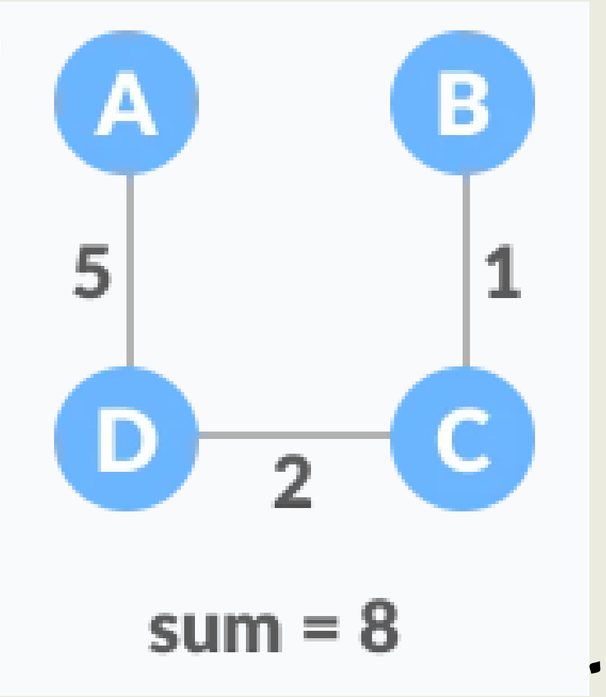


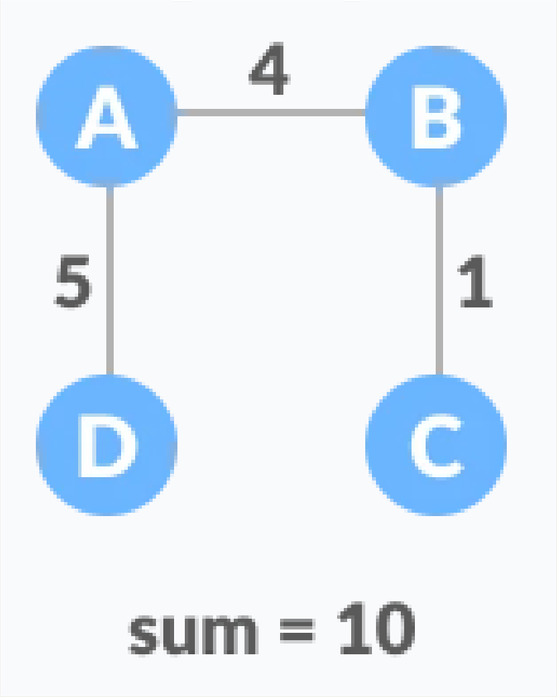
Example of a Spanning Tree

The initial graph is:

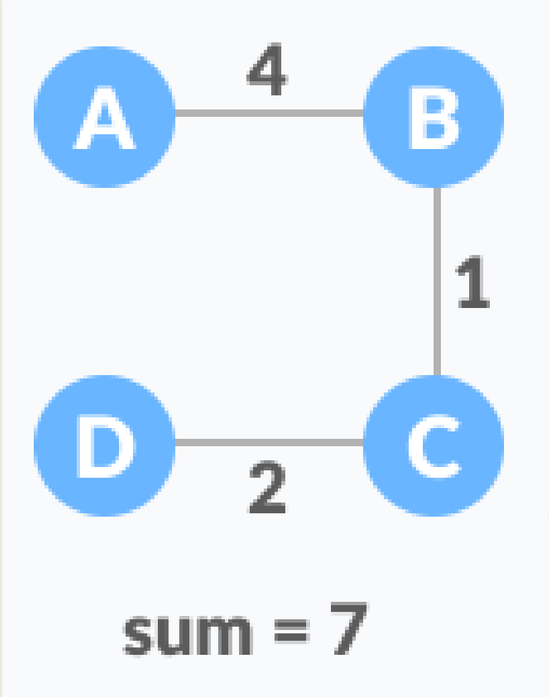
The possible spanning trees from the above graph are:







The minimum spanning tree from the above spanning trees is:



The minimum spanning tree from a graph is found using the algorithm of Prim’s and Kruskal’s Algorithm

Spanning Tree Applications

- Computer Network Routing Protocol
- Cluster Analysis
- Civil Network Planning

Minimum Spanning tree Applications

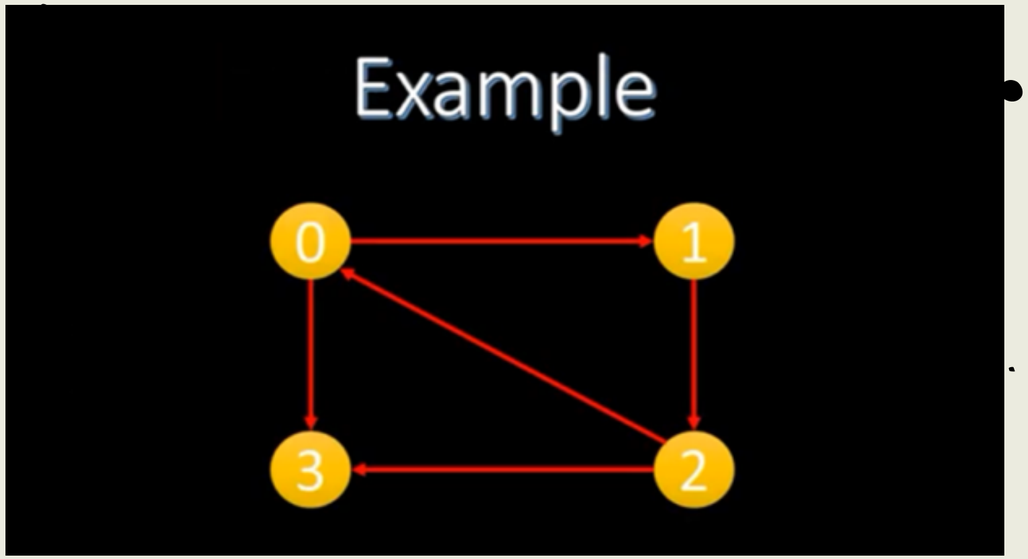
- To find paths in the map
- To design networks like telecommunication networks, water supply networks, and electrical grids.

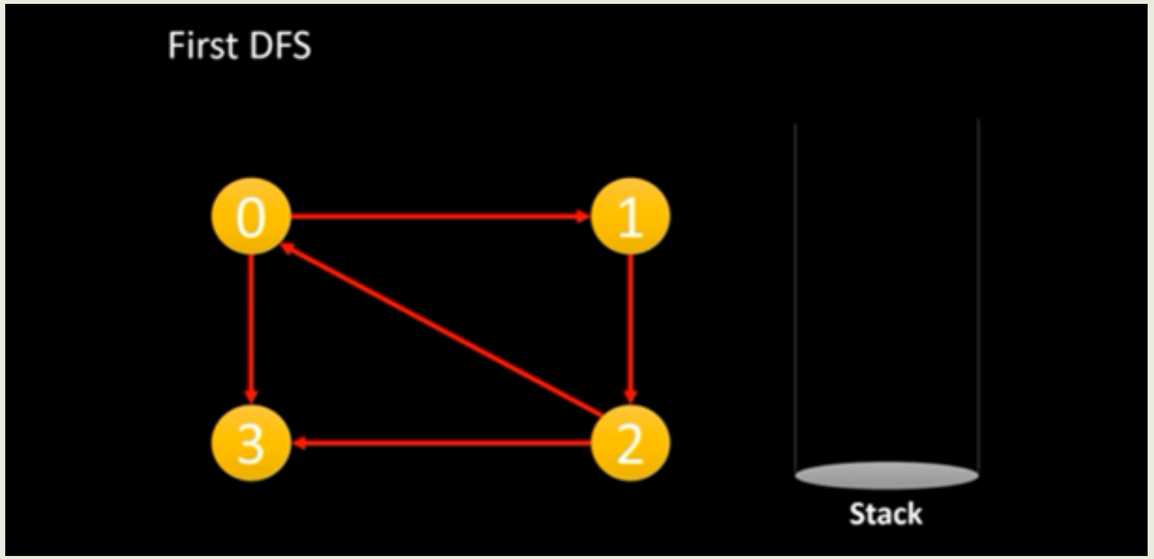
Strongly Connected Components

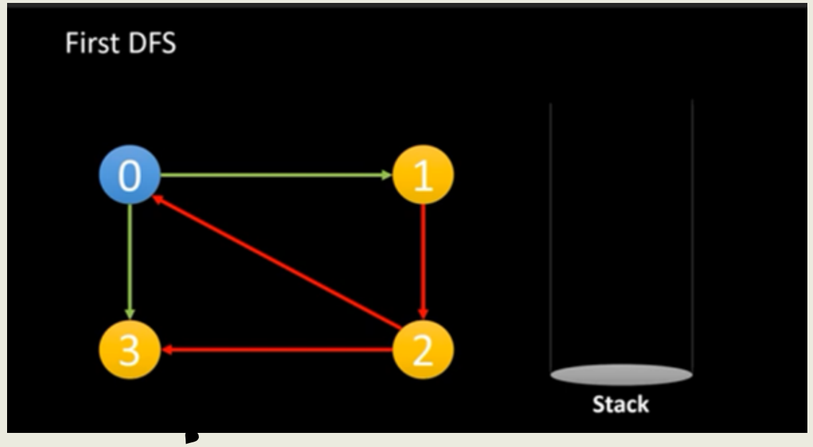
Are you ready?

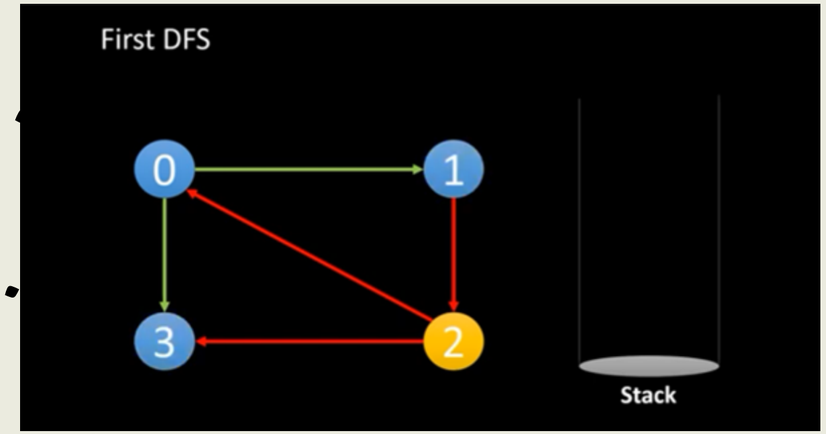
Strongly Connected Components

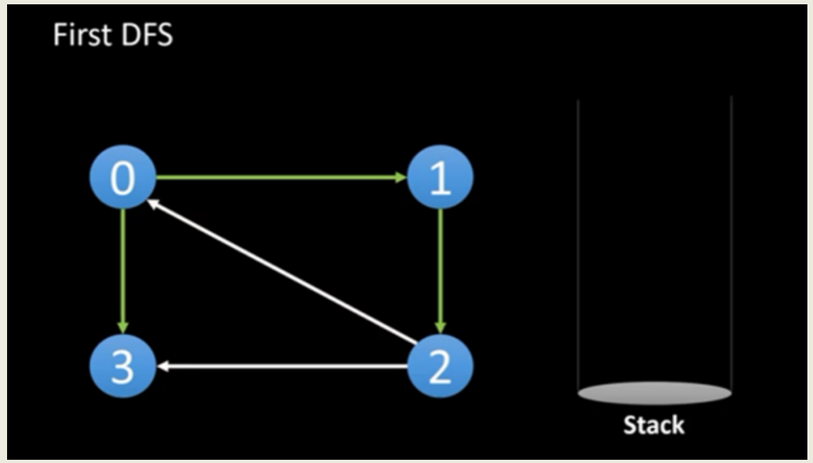
A strongly connected component is the portion of a directed graph in which there is a path from each vertex to another vertex. It is applicable only on a directed graph.

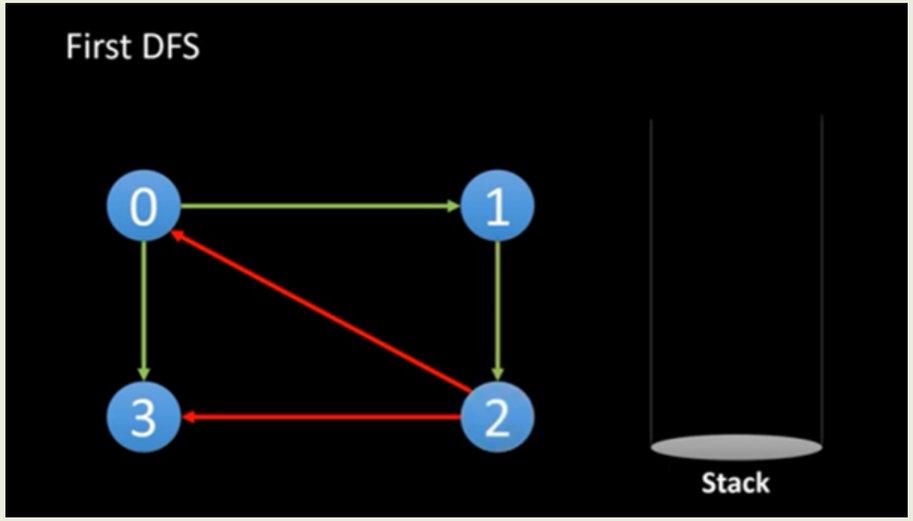


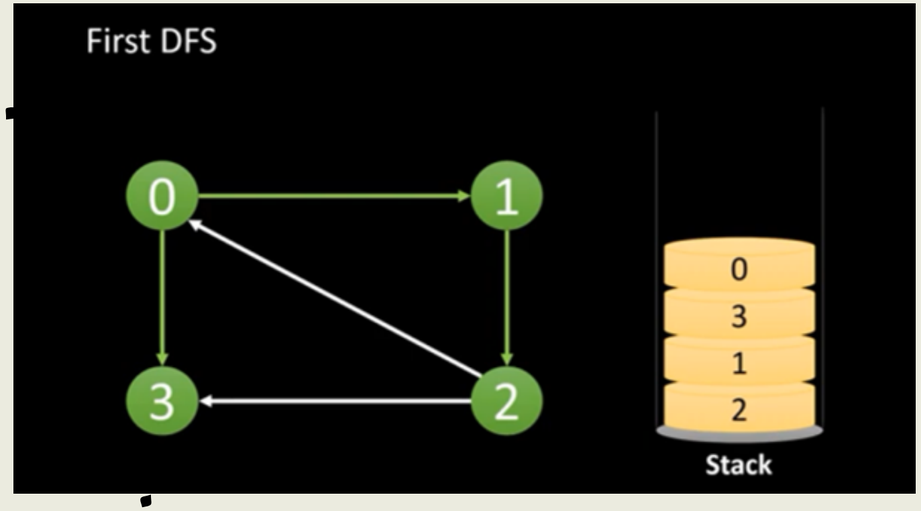


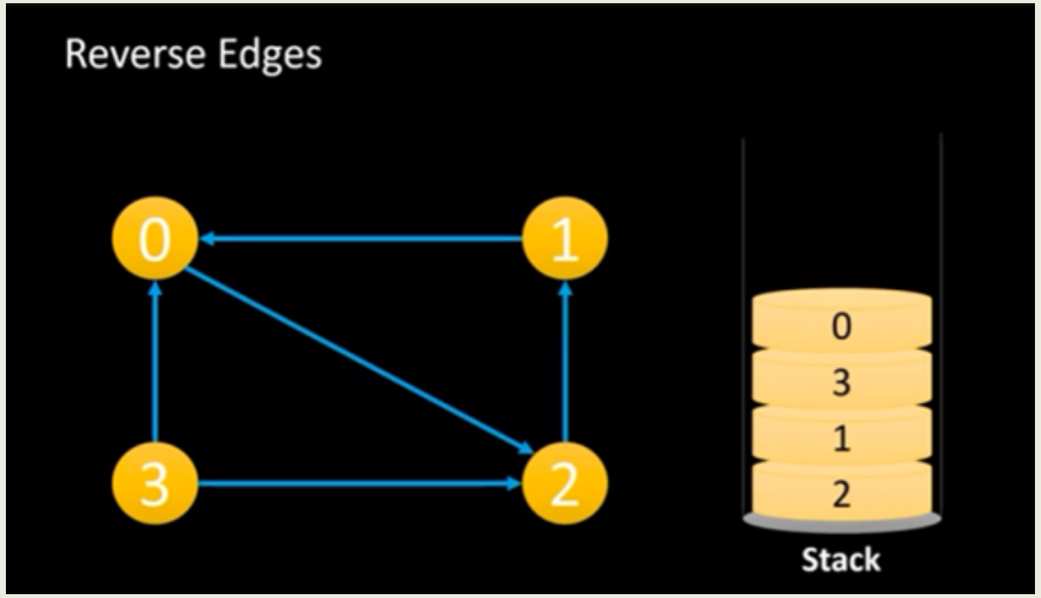

First DFS
0
1
3
2
Stack

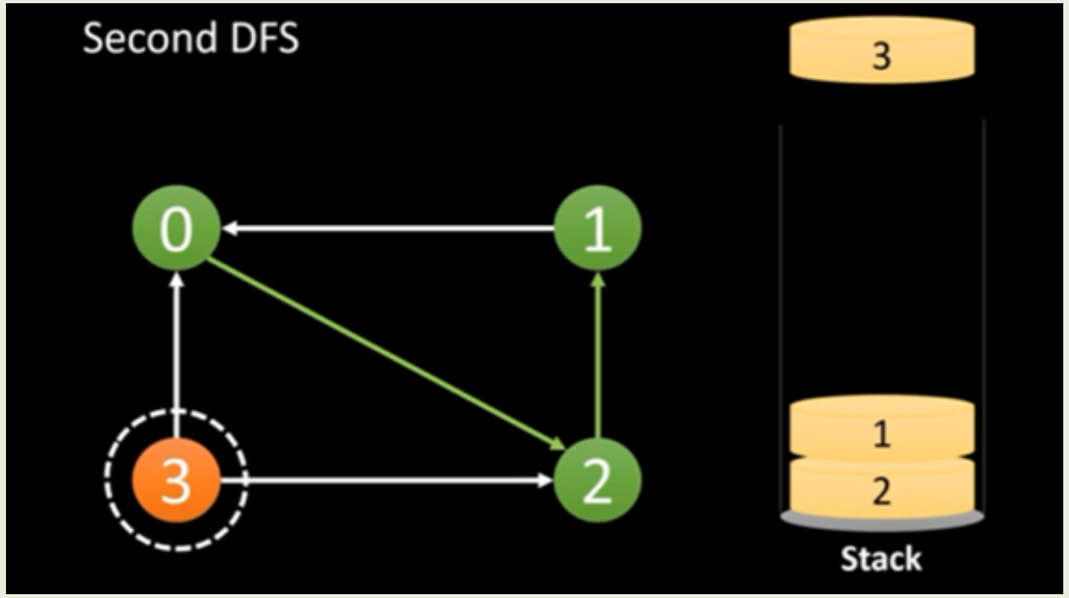

First DFS
0
1
3
2
Stack

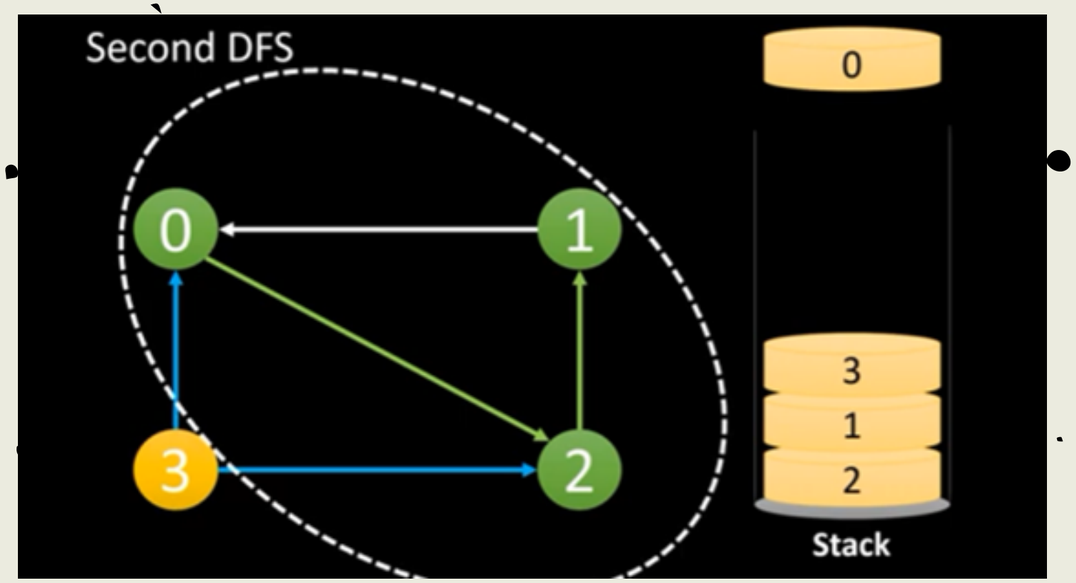

First DFS
0
1
3
2
Stack

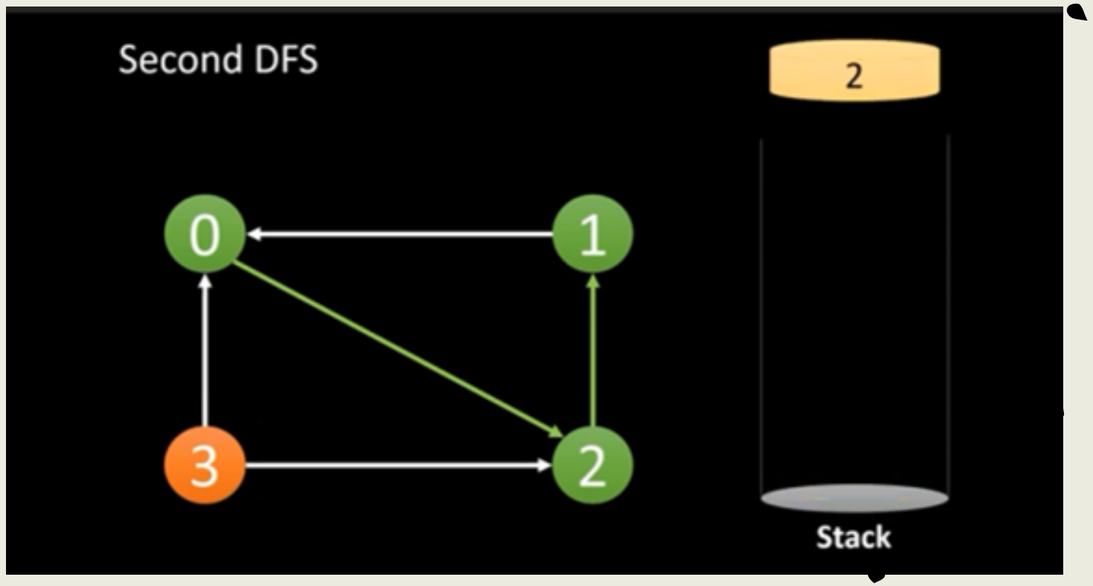

First DFS
0
1
3
2
Stack

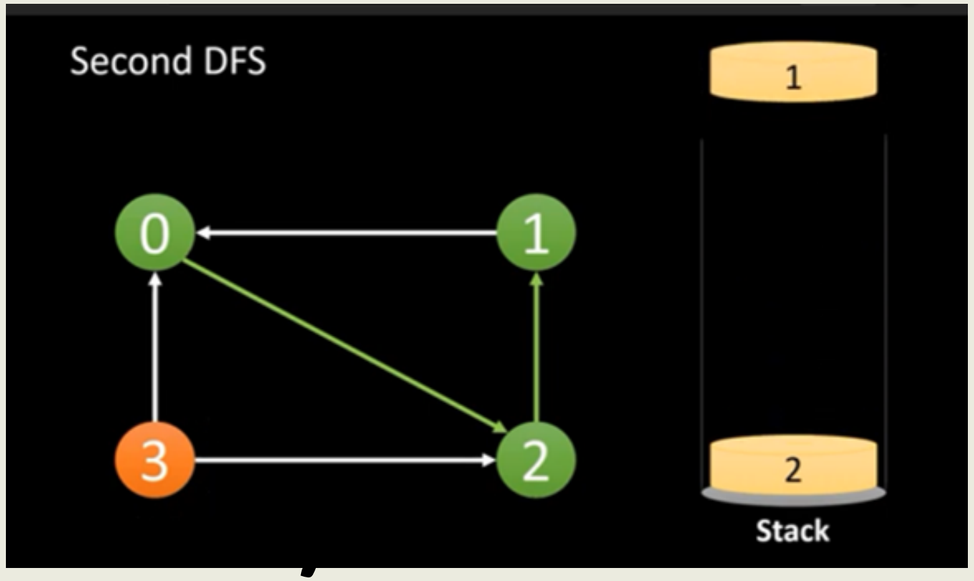

First DFS
0
1
3
2
0
3
1
2
Stack

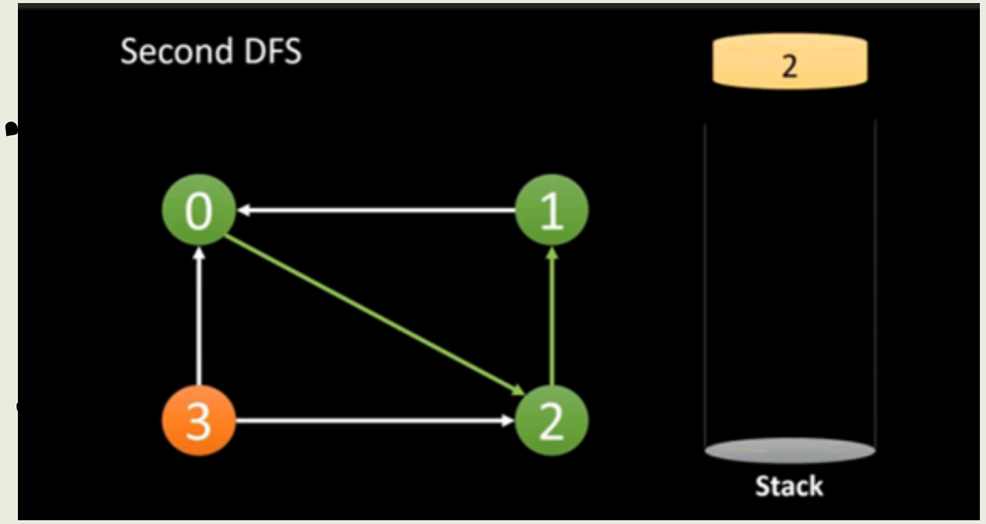

Reverse Edges
0
1
3
2
0
3
1
2
Stack

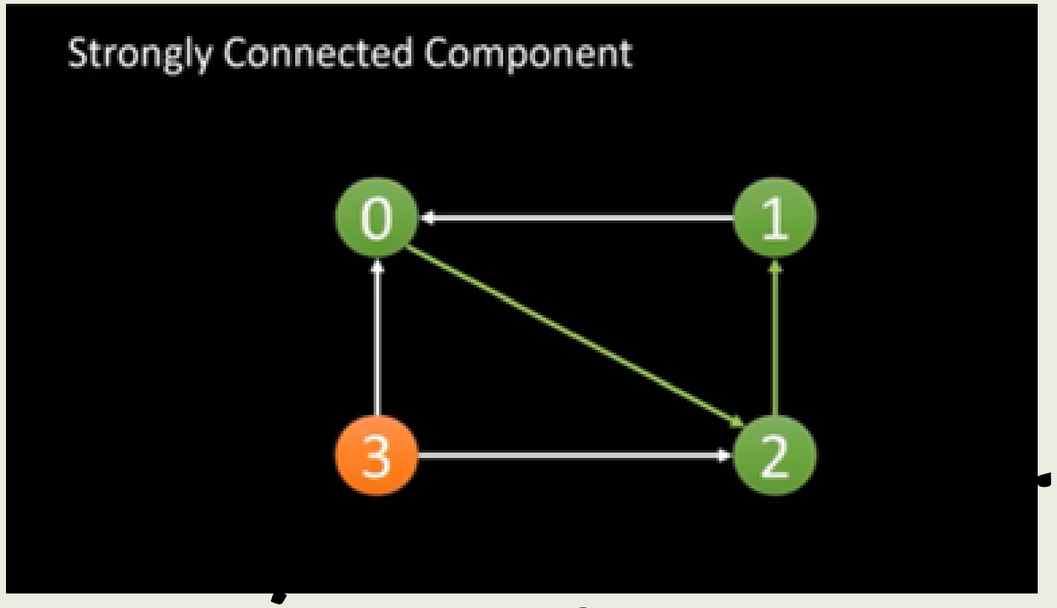

Second DFS
0
1
3
2
3
1
2
Stack


Second DFS
0
1
3
2
0
3
1
2
Stack


Second DFS
0
1
3
2
2
Stack


Second DFS
1
0
1
3
2
2
Stack


Second DFS
2
0
1
3
2
Stack

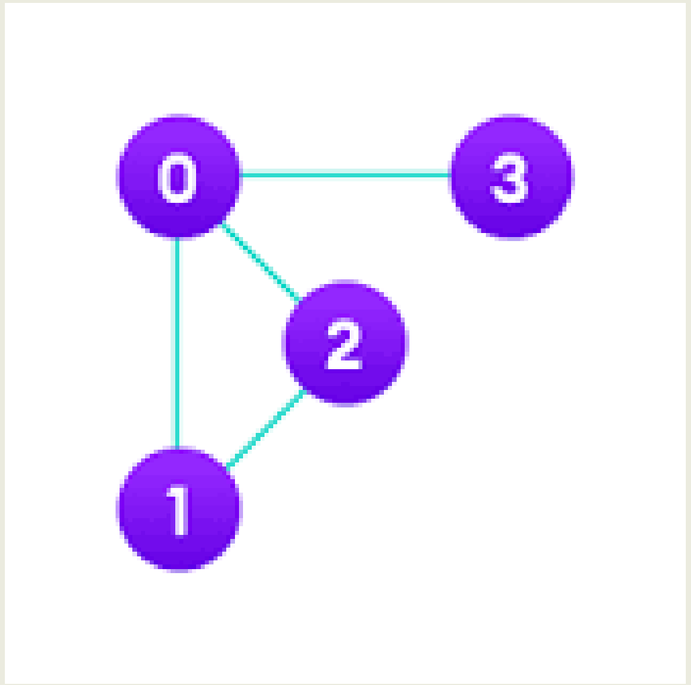

Strongly Connected Component
0
1
3
2

Adjacency Matrix

Are you ready?

Adjacency Matrix

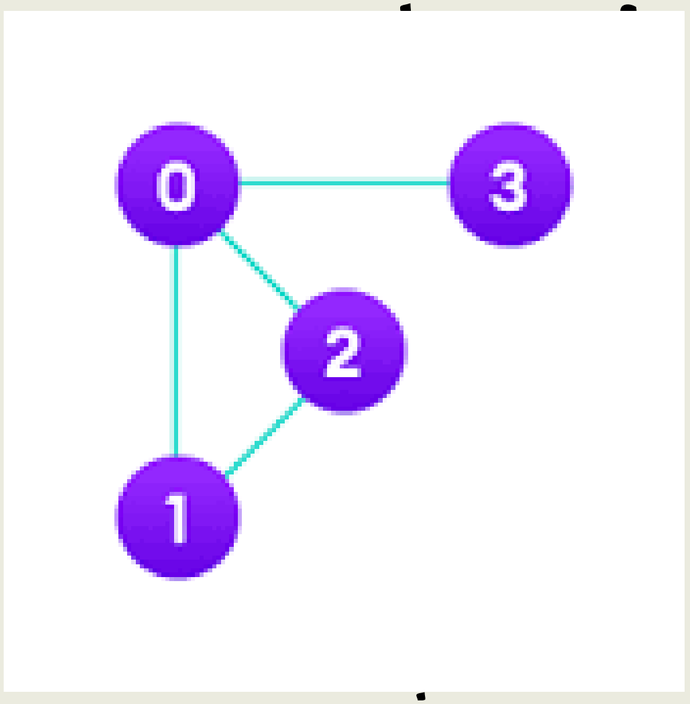
The adjacency matrix, sometimes also called the connection matrix, of a simple labeled graph is a matrix with rows and columns labeled by graph vertices, with a 1 or 0 in position according to whether and are adjacent or not. For a simple graph with no self-loops, the adjacency matrix must have 0s on the diagonal.



In graph theory, an adjacency matrix is nothing but a square matrix utilized to describe a finite graph. The components of the matrix express whether the pairs of a finite set of vertices (also called nodes) are adjacent in the graph or not.

$$Square\ Matrix\ M = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}$$

In the special case of a finite simple graph, the adjacency matrix is a (0,1)-matrix with zeros on its diagonal. If the graph is undirected (i.e. all of its edges are bidirectional), the adjacency matrix is symmetric.



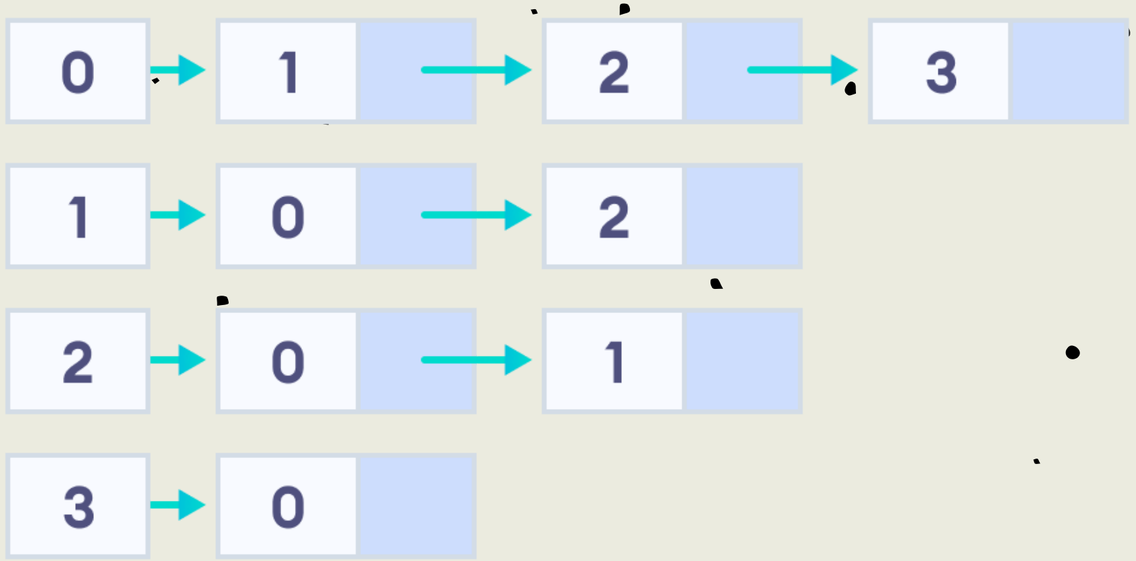
An adjacency matrix is a way of representing a graph as a matrix of booleans (0's and 1's). A finite graph can be represented in the form of a square matrix on a computer, where the boolean value of the matrix indicates if there is a direct path between two vertices.

Adjacency List

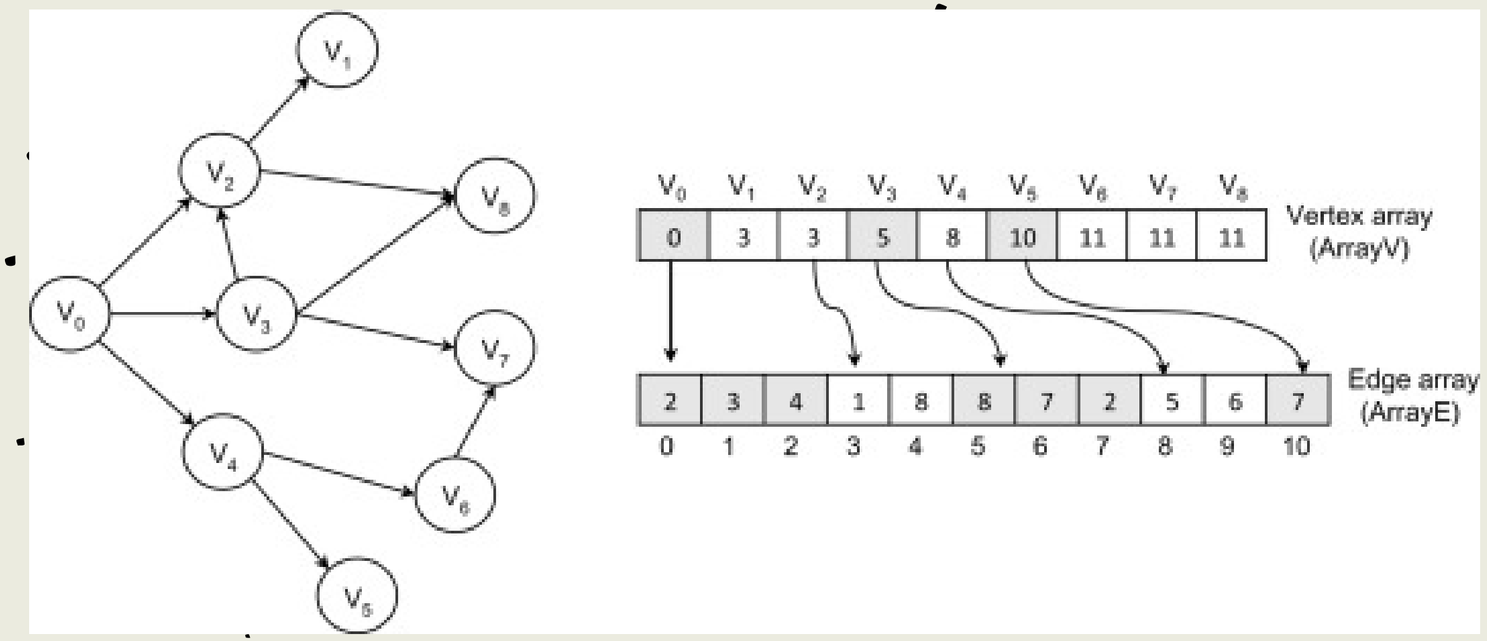
Are you ready?

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.



In general, an adjacency list consists of an array of vertices (ArrayV) and an array of edges (ArrayE), where each element in the vertex array stores the starting index (in the edge array) of the edges outgoing from each node. The edge array stores the destination vertices of each edge (Fig. 2). This allows visiting the neighbors of a vertex v by reading the edge array from ArrayV[v] to ArrayV[v + 1].



Pros of Adjacency List

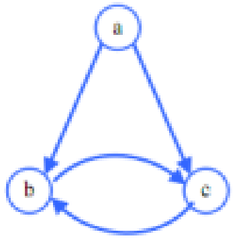
An adjacency list is efficient in terms of storage because we only need to store the values for the edges. For a sparse graph with millions of vertices and edges, this can mean a lot of saved space. It also helps to find all the vertices adjacent to a vertex easily.

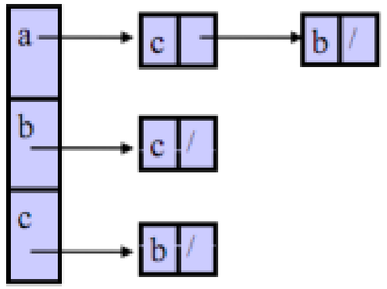
Cons of Adjacency List

Finding the adjacent list is not quicker than the adjacency matrix because all the connected nodes must be first explored to find them.

Adjacency List

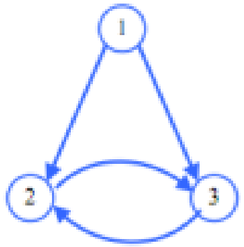
An adjacency list is a list of lists. Each list corresponds to a vertex u and contains a list of edges (u, v) that originate from u. Thus, an adjacency list takes up $\Theta(V + E)$ space.





Adjacency Matrix

An adjacency matrix is a $|V| \times |V|$ matrix of bits where element (i, j) is 1 if and only if the edge (v_i, v_j) is in E. Thus an adjacency matrix takes up $\Theta(|V|^2)$ storage (note that the constant factor here is small since each entry in the matrix is just a bit).



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

Depth First Search (DFS)

Are you ready?

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:

1. Visited
2. Not Visited

The DFS algorithm works as follows:

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

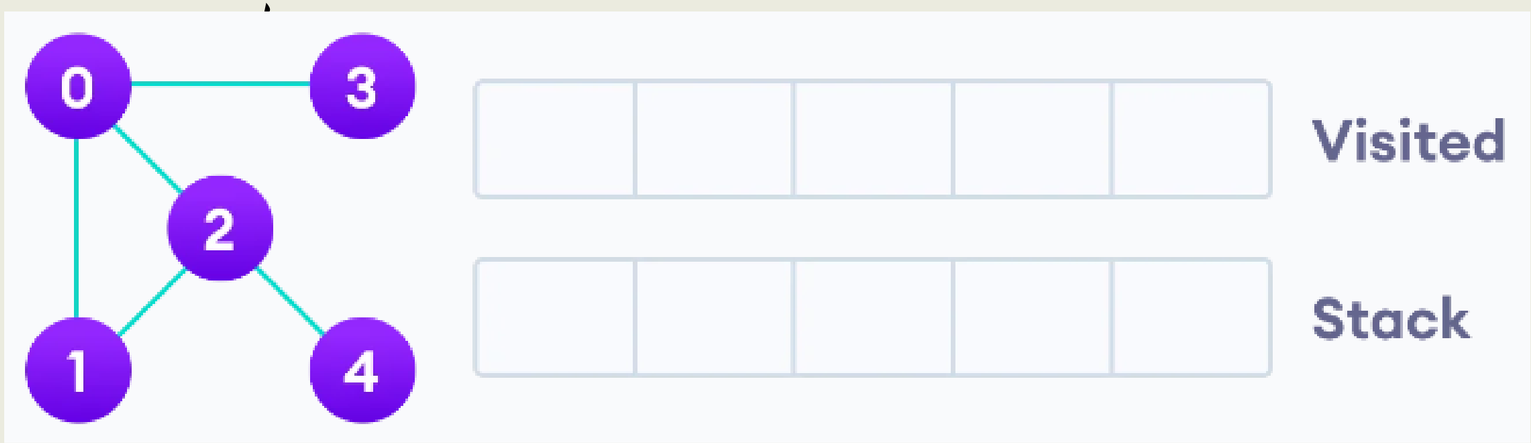
2. Take the top item of the stack and add it to the visited list.

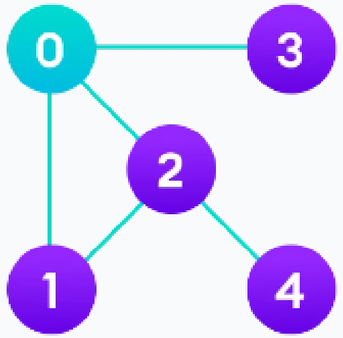
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't the visited list to the top of the stack.

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

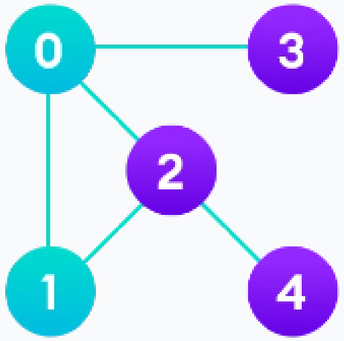
Depth First Search Example

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

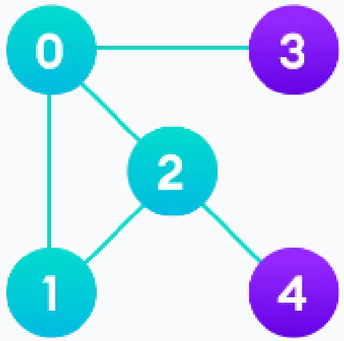




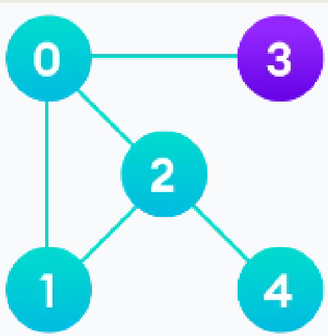
0					Visited
1	2	3			Stack



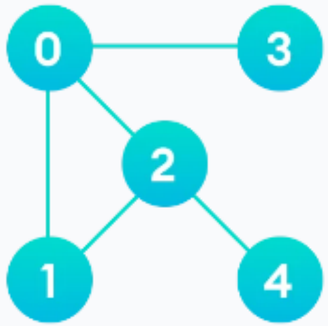
0	1				Visited
2	3				Stack



0	1	2			Visited
4	3				Stack



0	1	2	4		Visited
3					Stack



0	1	2	4	3	Visited
					Stack

Application of DFS Algorithm

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

Breadth First Search (BFS)

Are you ready?

Breadth First Search (BFS)

Traversal means visiting all the nodes of a graph. 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

BFS Algorithm A standard BFS implementation puts each vertex of the graph into one of two categories:

1. Visited

2. Not Visited

The algorithm works as follows:

1. Start by putting any one of the graph's vertices at the back of a queue.

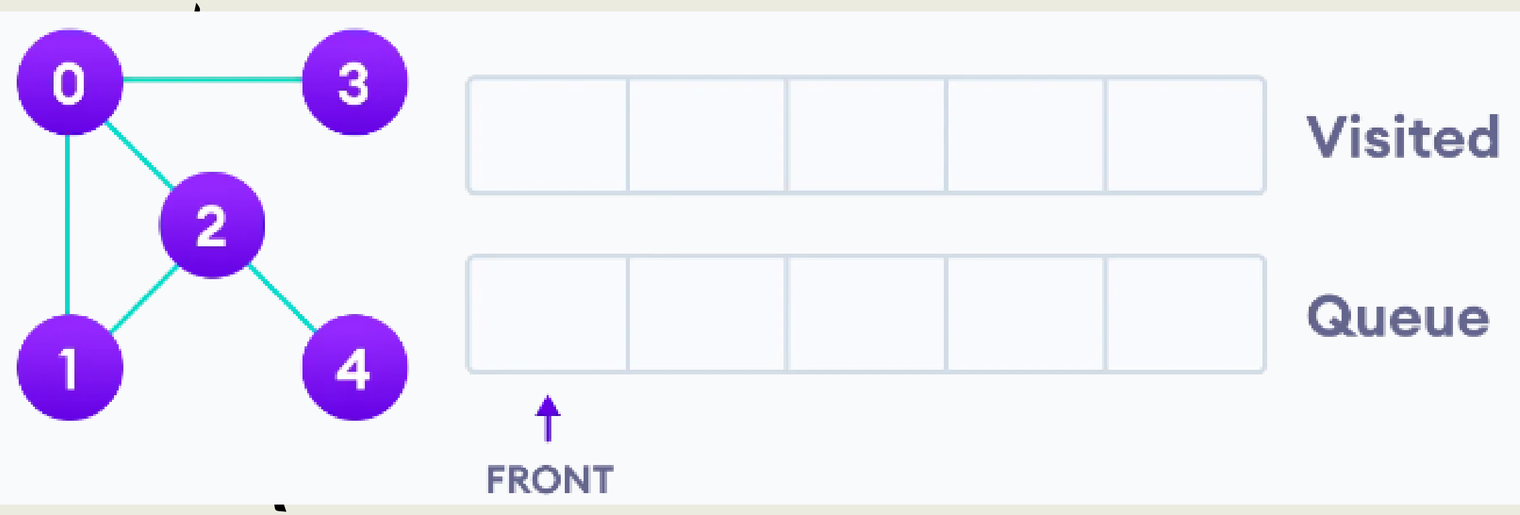
2. Take the front item of the queue and add it to the visited list.

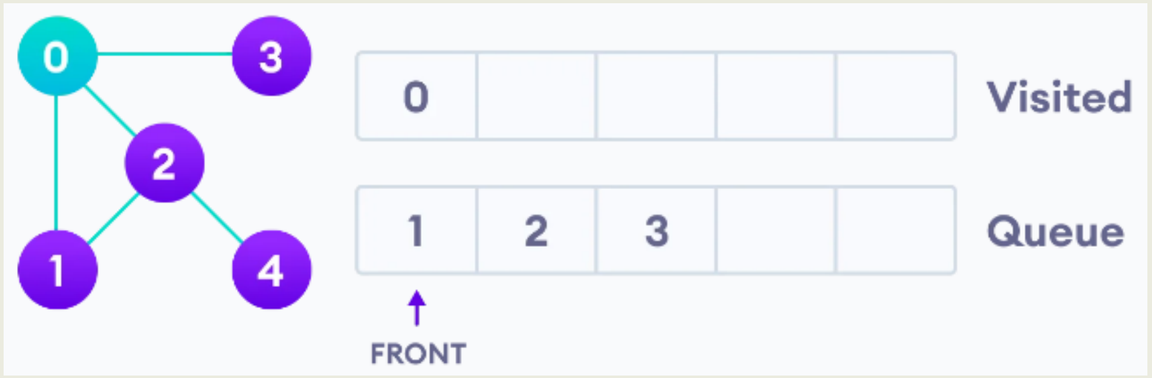
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.

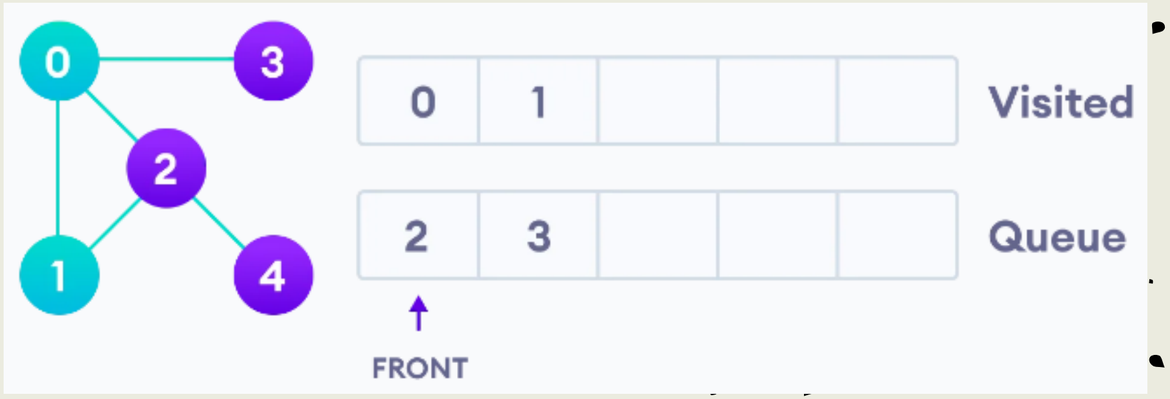
4. Keep repeating steps 2 and 3 until the queue is empty.

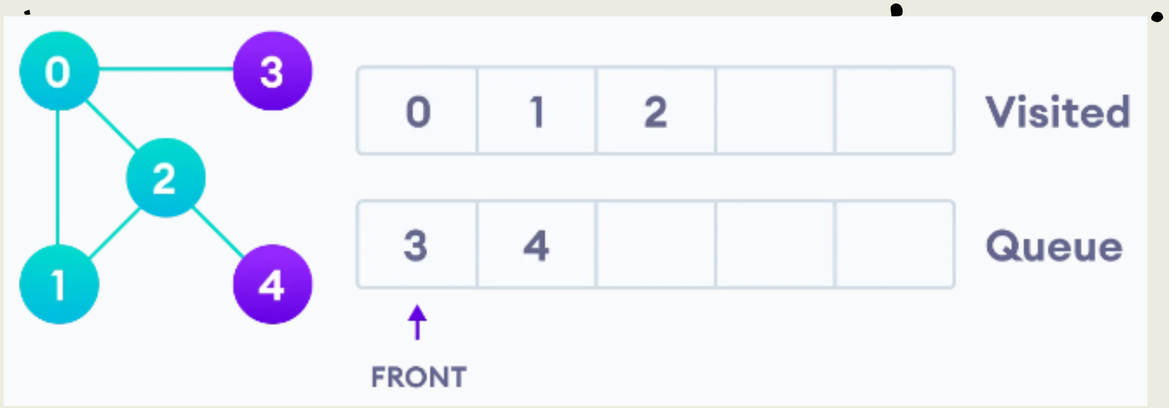
BFS Example

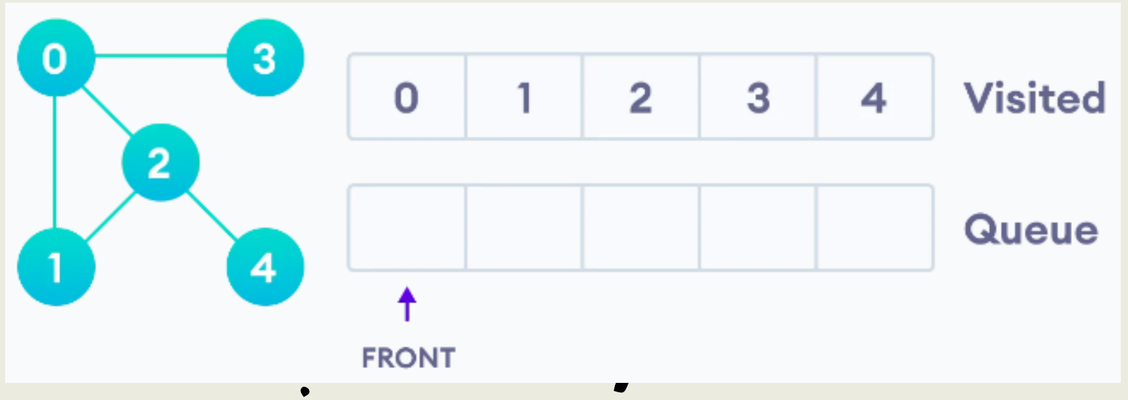
Let’s see how the Breadth First Search algorithm works with an example. We use an undirected graph with 5 vertices.




0
3
2
1
4
0
Visited
1
2
3
Queue
FRONT


0
3
2
1
4
0
1
Visited
2
3
Queue
FRONT


0
3
2
1
4
0
1
2
Visited
3
4
Queue
FRONT



BFS Algorithm Applications

1. To build index by search index
2. For GPS navigation
3. Path finding algorithms
4. In Ford-Fulkerson algorithm to find maximum flow in a network
5. Cycle detection in an undirected graph
6. In minimum spanning tree

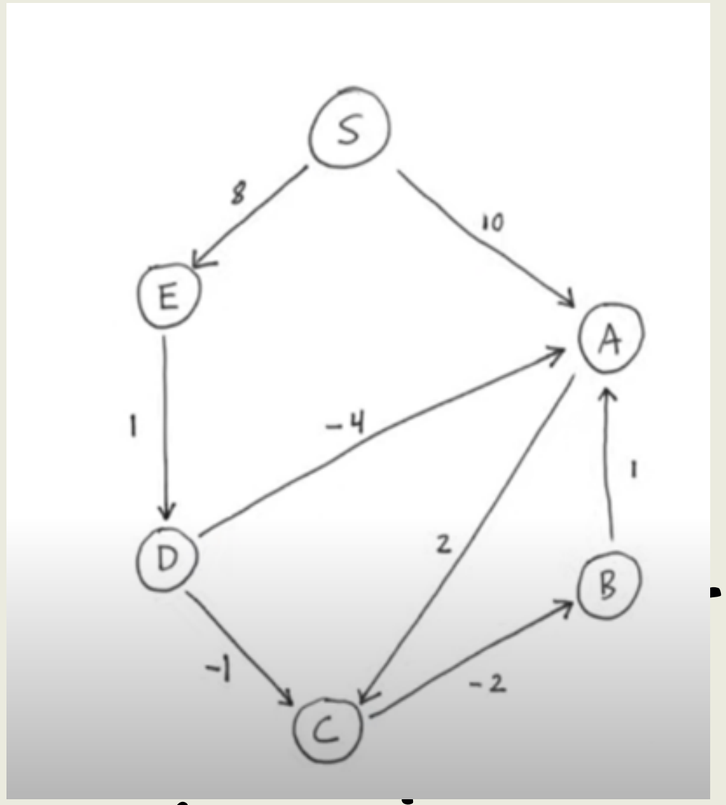
Bellman Ford's Algorithm

Are you ready?

Bellman Ford's Algorithm

The Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers.

Example



S	A	B	C	D	E
0	∞	∞	∞	∞	∞
0	10	10	12	9	8
0	5	10	8	9	8
0	5	5	7	9	8
0	5	5	7	9	8

Thank you!

Have a great day ahead.