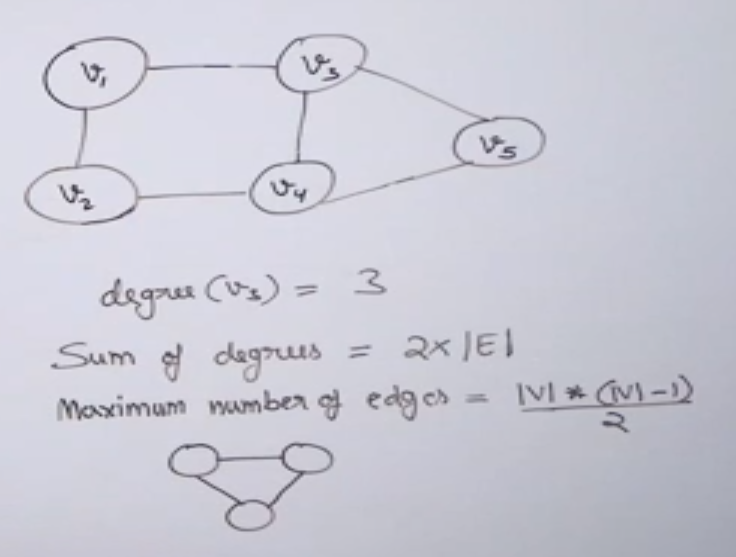
# Introduction

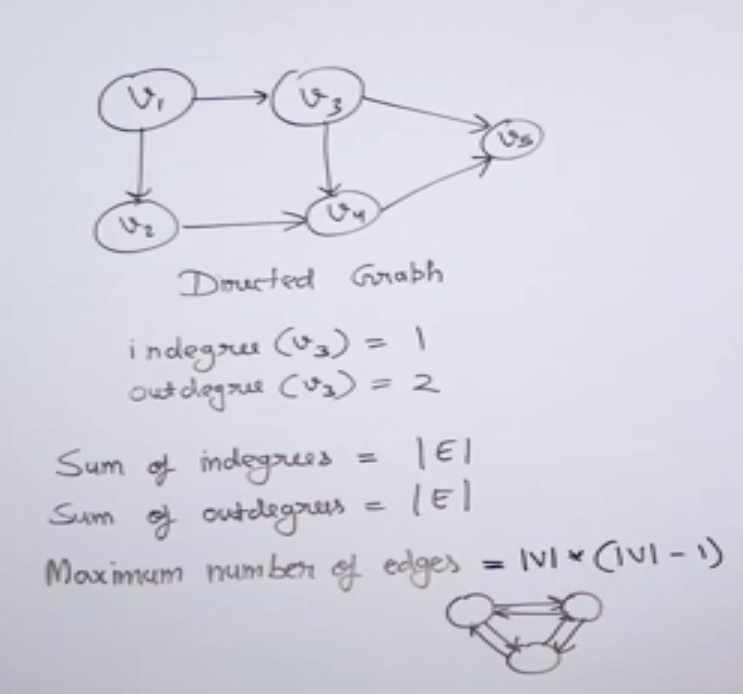
A ***Graph*** is a data structure that consists of the following two components:

1. A finite set of vertices also called nodes.
2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not the same as (v, u) in case of a directed graph(digraph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

## Degree of Graph:

1. Degree of undirected graph:



1. Degree of directed graph:  
   

In general, number of edges,

Undirected graph:

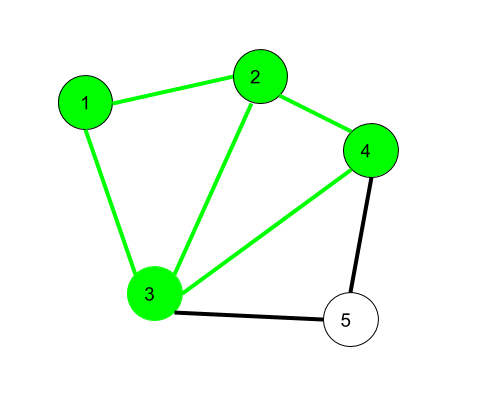
Directed graph:

# Walks, Circuits, Paths and Cycles in Graph

## Walk

A walk is a sequence of vertices and edges of a graph i.e. if we traverse a graph then we get a walk.

* Vertex can be repeated
* Edges can be repeated
* Walk can repeat anything (edges or vertices).



**Here 1->2->3->4->2->1->3 is a walk**

**Open walk-**A walk is said to be an open walk if the starting and ending vertices are different i.e. the origin vertex and terminal vertex are different.

1->2->3->4->5->3-> is an open walk.

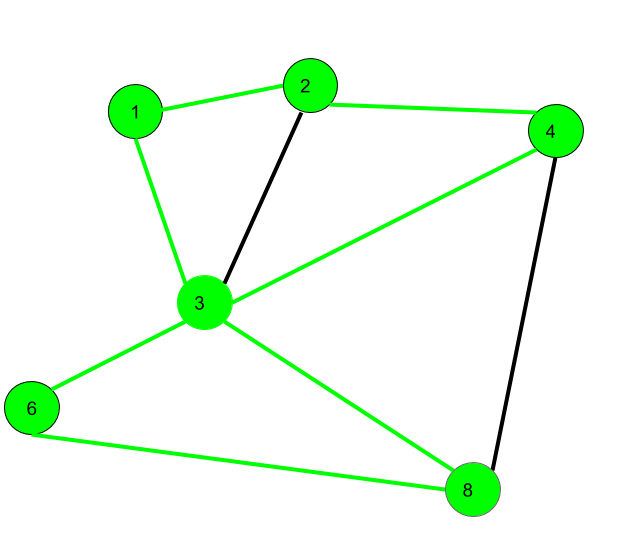
**Closed walk-**A walk is said to be a closed walk if the starting and ending vertices are identical i.e. if a walk starts and ends at the same vertex, then it is said to be a closed walk.

1->2->3->4->5->3->1-> is a closed walk.

## Circuit

Traversing a graph such that not an edge is repeated but vertex can be repeated.

* Vertex can be repeated
* Edge not repeated

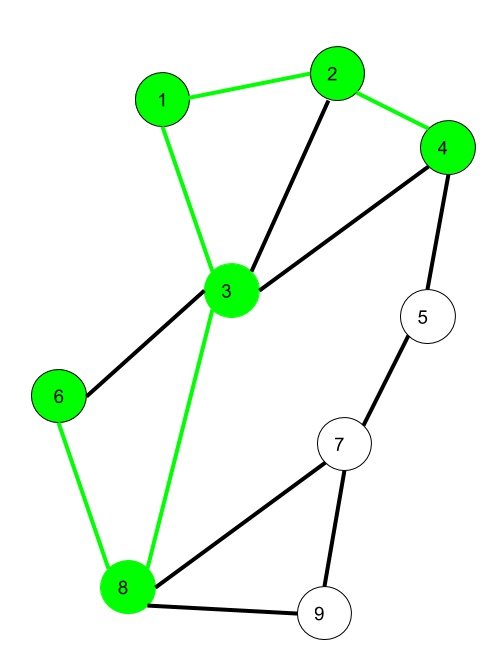


Here 1->2->4->3->6->8->3->1 is a circuit

## Path

Neither vertices nor edges are repeated i.e. if we traverse a graph such that we do not repeat a vertex and nor we repeat an edge.

* It is also an open walk
* Vertex not repeated
* Edge not repeated

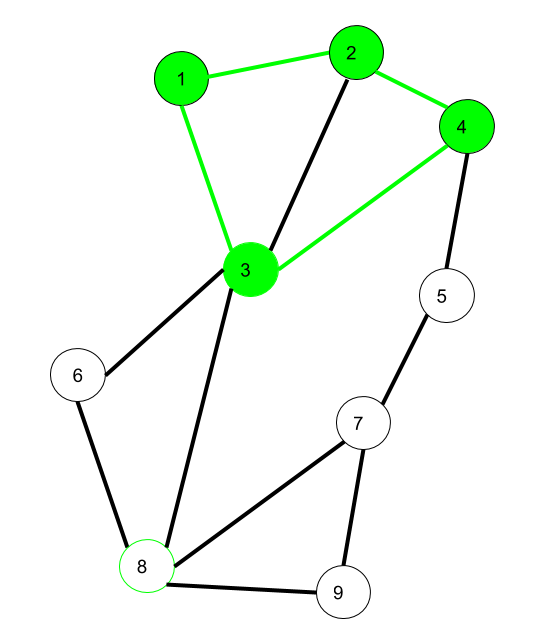


Here 6->8->3->1->2->4 is a Path

## Cycle

Traversing a graph such that we do not repeat a vertex nor we repeat a edge but the starting and ending vertex must be same i.e. we can repeat starting and ending vertex only then we get a cycle.

* Vertex not repeated
* Edge not repeated



Here 1->2->4->3->1 is a cycle.

# Graph Representation

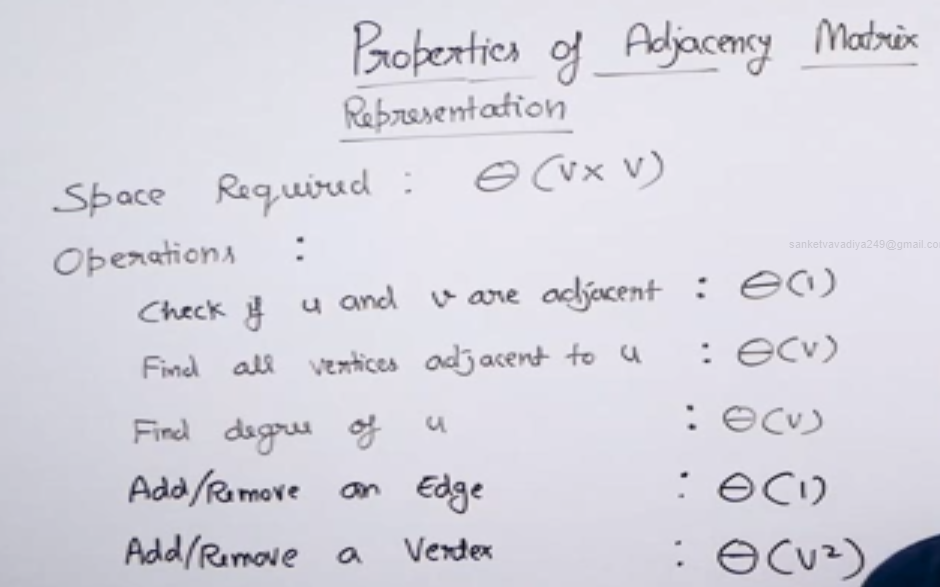
## Adjacency matrix

The Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph.

Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j.

Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs.

If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.



To add/remove vertex from graph takes time. In both the we first need to create new matrix and then copy all the element of matrix to new matrix.

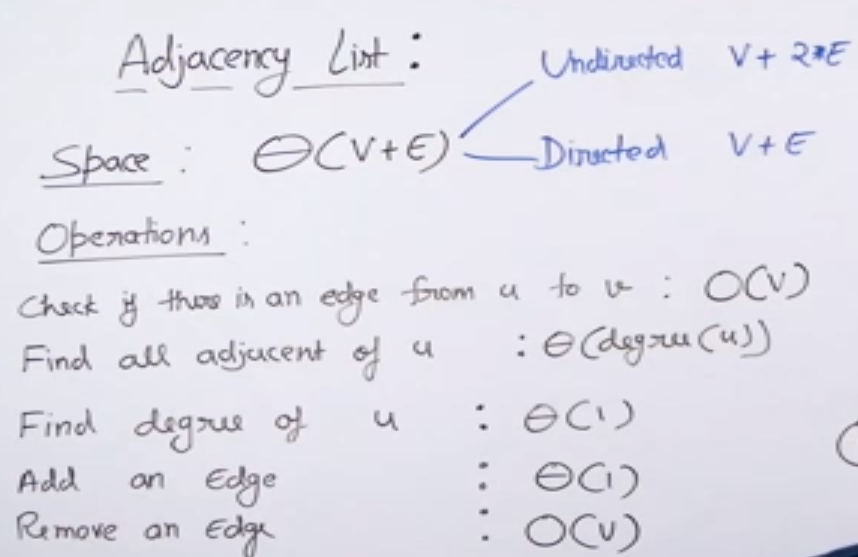
***Pros***: Representation is easier to implement and follow. Removing an edge takes O(1) time. Queries like whether there is an edge from vertex 'u' to vertex 'v' are efficient and can be done O(1).  
  
***Cons***: Consumes more space O(V^2). Even if the graph is sparse(contains less number of edges), it consumes the same space. Adding a vertex is O(V^2) time.

## Adjacency list:

Graph can also be implemented using an array of lists.

That is every index of the array will contain a complete list. Size of the array is equal to the number of vertices and every index **i** in the array will store the list of vertices connected to the vertex numbered i. Let the array be array[].

An entry array[i] represents the list of vertices adjacent to the**i**th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be represented as lists of pairs. Following is the adjacency list representation of the above example undirected graph.



***Pros***: Saves space O(|V|+|E|). In the worst case, there can be C(V, 2) number of edges in a graph thus consuming O(V^2) space. Adding a vertex is easier.  
  
***Cons***: Queries like whether there is an edge from vertex u to vertex v are not efficient and can be done O(V).

## Comparison of adjacency list and matrix:

