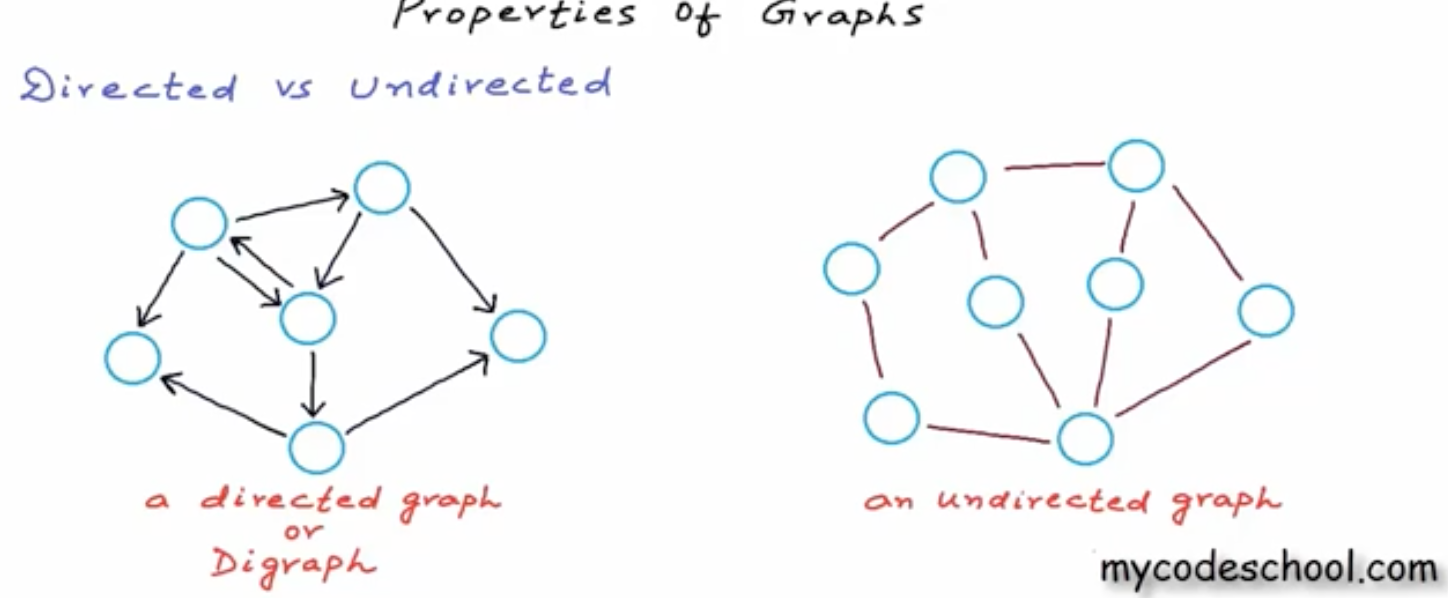
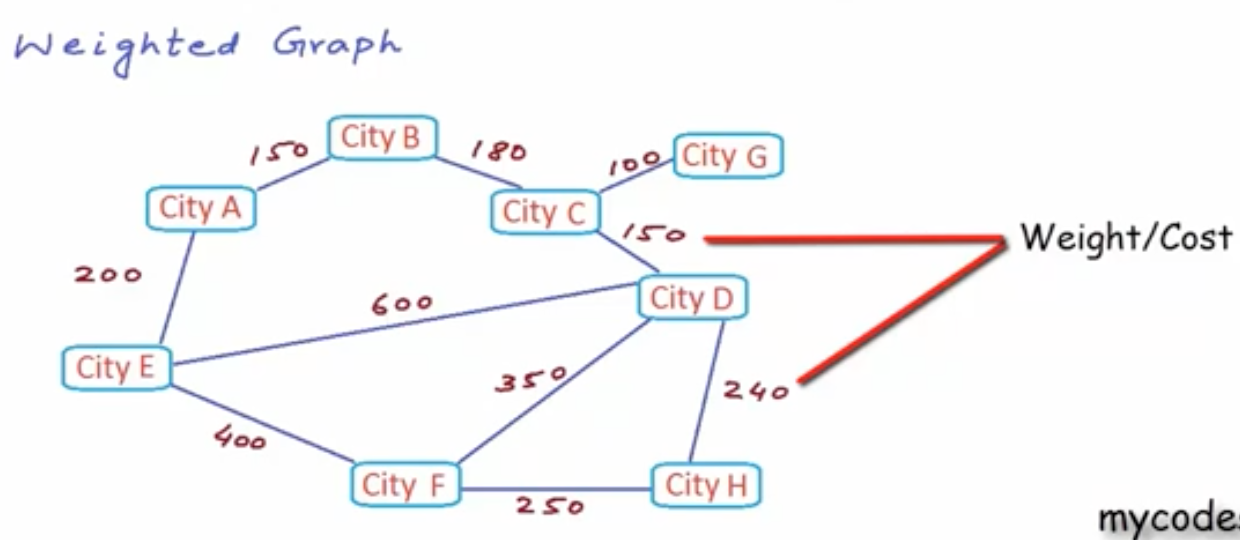
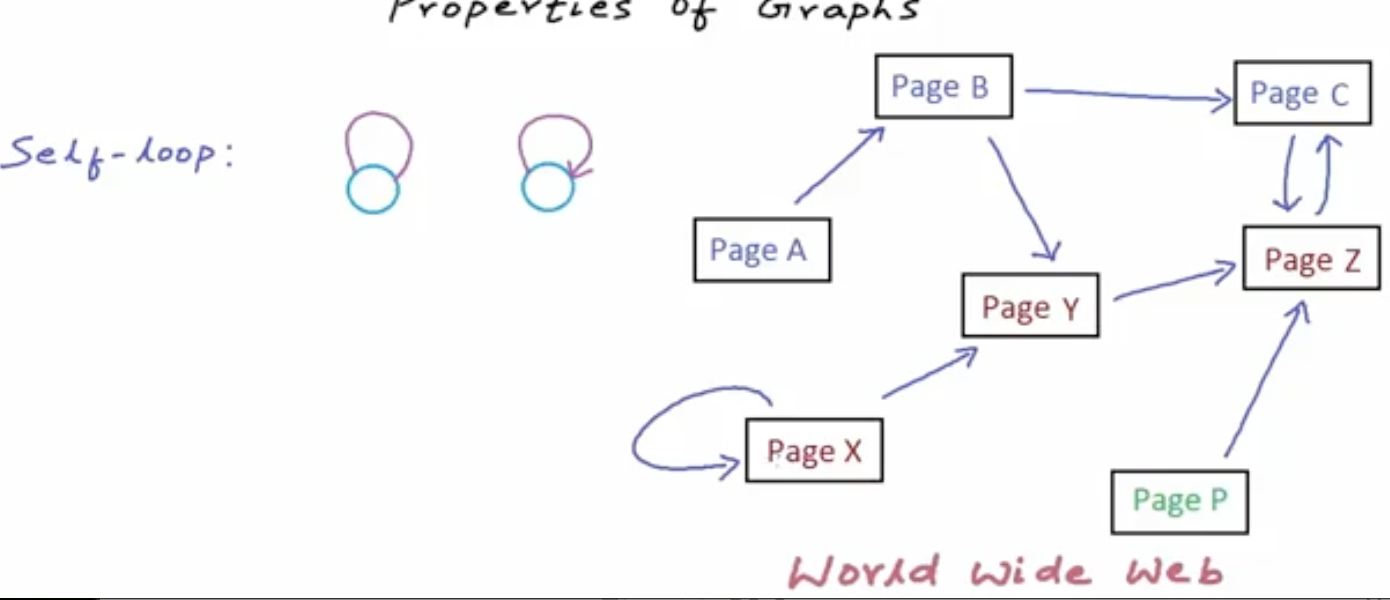
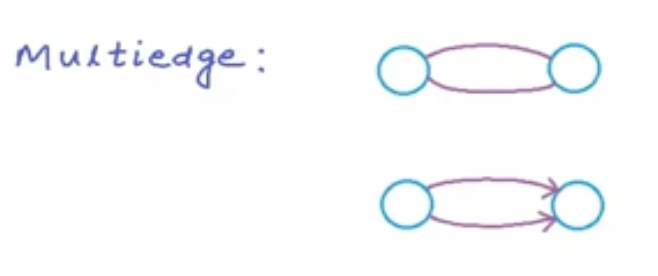
# Graph







e.g. A webpage can have a link to itself.



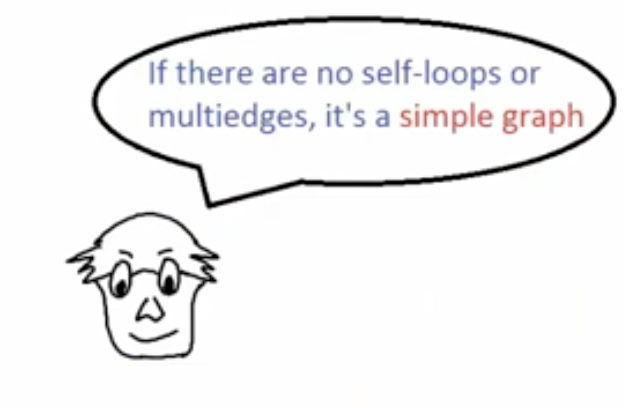
There can be more than one edges between vertices. E.g. there can be multiple paths for flights to go from one city to another.



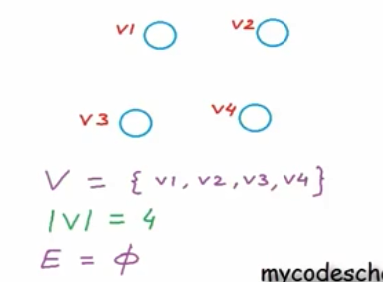
A graph with self-loop or multi-edge is difficult to solve, so they can also be said as complex graphs.

## Simple Graph

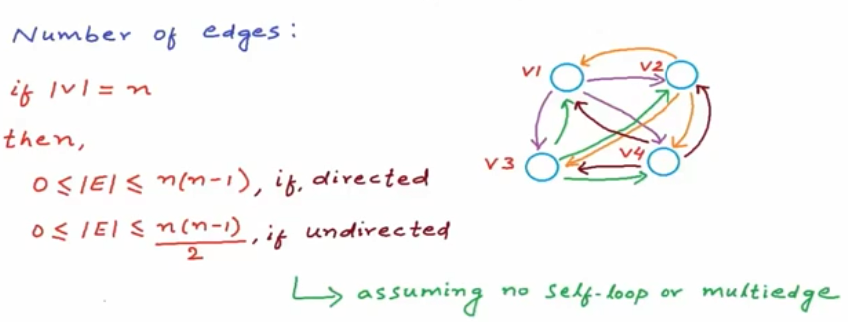
A graph without self-loop or multi-edge is called a simple graph.

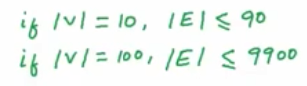


Minimum number of edges in a simple graph can be 0.



Max number of edges in a simple directed graph can be v(v-1) where v is a number of vertices and in undirected graph it will be half of it v(v-1)/2

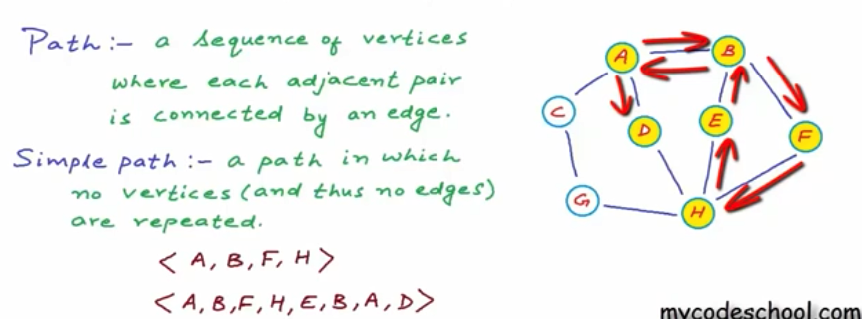


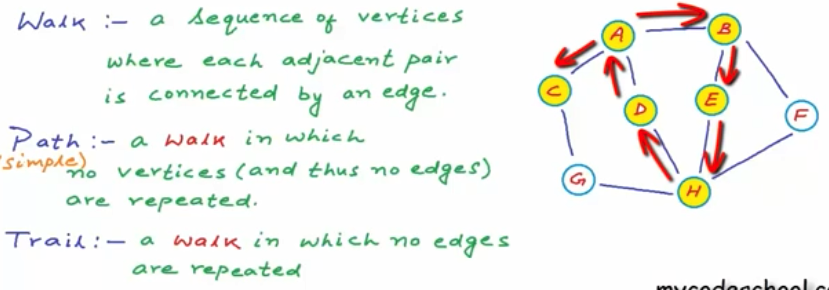


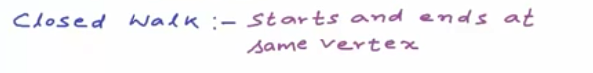
As you see, as number of vertices grow, number of edges grow tremendously (close to square of number of vertices). This kind of graph is called a Dense Graph, where number of vertices are close to max. In real life, Dense graph is very rare. If number of edges are close to number of vertices, then that is called a Sparse Graph. There is no specific limit on number of edges to call a graph dense or sparse. It all depends on the context. A lot of decisions are made based on the whether the graph is dense or sparse.

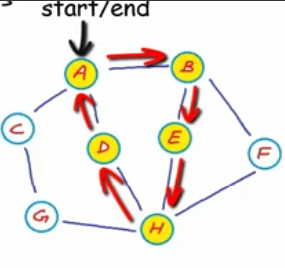
We store Dense Graph in Adjacency Matrix and Sparse Graph in Adjacency List. We will see the difference between these two storage styles later on.

## Path, Walk



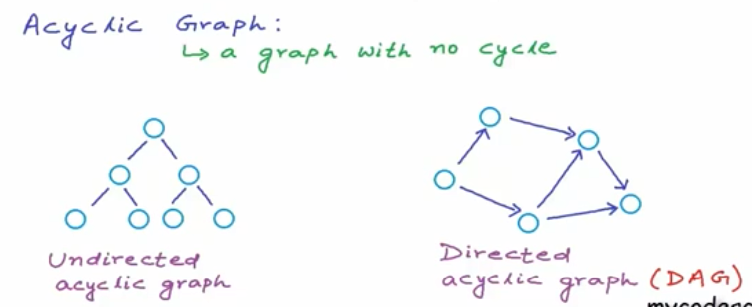




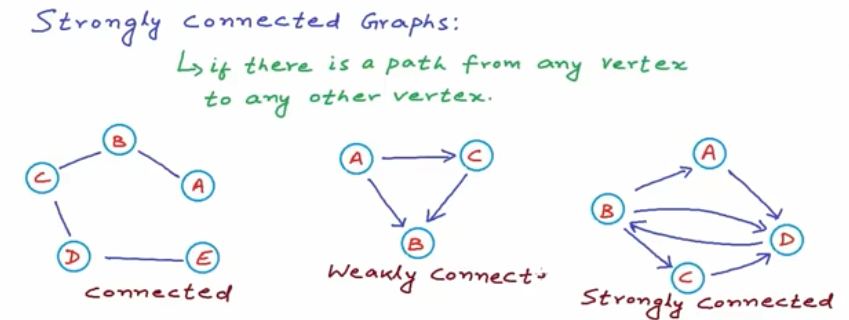


## Cycle





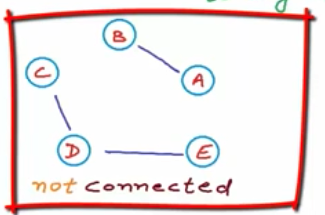
## Connected, Strongly Connected, Weekly Connected Graphs



If you can reach from any vertex to any other vertex in Undirected Graph, the it is called a connected graph.

If you can do the same in Directed Graph, then it is called Strongly Connected Graph.

If by converting Directed Graph into Undirected one, you can convert into Connected Graph, then that graph is called Weakly Connected Graph.



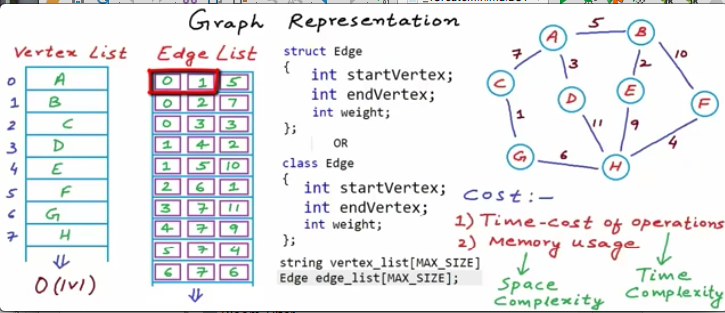
## Graph Storage

To store vertices, you need Vertex List.

To store edges, you have 3 options.

* [Edge List](#_Edge_List) (not preferable due to very high time complexity)
* [Adjacency Matrix](#_Adjacency_Matrix_(Ideal) (Ideal for Dense Graph)
* [Adjacency List](#_Adjacency_List_(Ideal_1) (Ideal for Sparse Graph)

### Edge List



Edge List contains references of list indices from Vertex List and a weight between those two vertices.

If we want to find all neighbors (adjacent vertices) of a specific vertex, you need to do a linear search (scan entire Edge List) on Edge List.

In Dense graph, number of edges can be close to v^2. It means that size of Edge List will be that much and so time complexity to iterate an Edge List will be that much.

**Edge List is not an efficient data structure to store a Graph. Do not use it.**

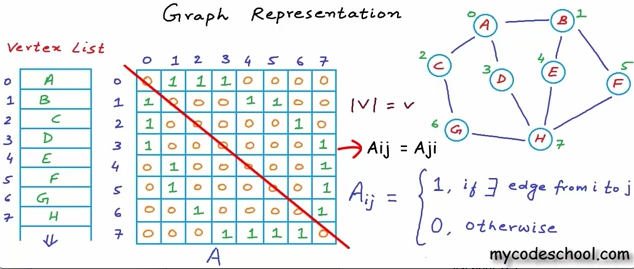
You can use Hash Table also to represent an Edge List. Grokking Algorithm book has examples with hash table for representing a graph.

### Adjacency Matrix (Ideal for Dense Graph)

You can use a matrix (double dimension array) to represent edges.

#### Symmetric Matrix

When you represent undirected edges, a[i][j] is same as a[j][i]. It is called a symmetric matrix. So, actually you need to traverse only half of the matrix. Another half is just a waste of memory.



If you need to find neighbors (adjacent vertices) of a vertex in a matrix, you just need to scan a related row in a matrix. E.g. to find neighbors of a node F, you need to scan row 5 in a matrix. This scanning will take O(V) time. A lot better than O(V^2) in Edge List.

To find out whether one vertex is a neighbor or not, it takes a constant time O(1), which is very efficient. A lot better than O(V^2) in Edge List.

Disadvantage:

Space complexity of a matrix is O(V^2).

Real life graphs are mostly sparse graphs. There will be lot of 0s in a matrix for a Sparse Graph. A lot of memory will be wasted. Good fit for a Sparse Graph is [Adjacency List](#_Adjacency_List_(Ideal).

#### Asymmetric Matrix

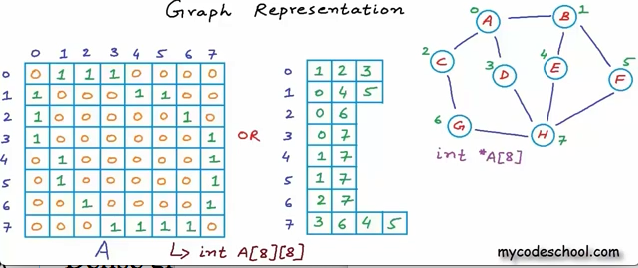
Matrix for a directed graph will be Asymmetric.

### Adjacency List (Ideal for Sparse Graph)

If you take out cells representing 0s from Adjacency Matrix, then that becomes Adjacency List.

To represent Adjacency List, you can have an array of size same as number of vertices and each array element will contain a linked list/binary search tree to store the edge information.

This looks like a hash table with collision scenario.



If you use linked list, to find out whether two vertices are neighbors of each other, it will take at the most O(V) time. If you use BST, it will take O(log V) time.

This is an ideal data structure to represent a Sparse Graph. Most real life graphs are Sparse Graphs.

Class Graph {

Vertex[] vertices;

LinkedList[] adjacentList;

}



## Graph Traversal Algorithms

BFS, DFS, Dikjstra’s Algorithm, BELLMAN-FORD's algorithm

Read from MyJavaProject/algorithms/treeandgraph/graph/

BfsDfsFromGrokkingAlgorithmBook.java

DijkstraAlgorithmForPositivelyWeightedGraphGrokkingAlgorithmBook.java