# sets & graphs

# Set

- Unordered collection of elements.
  - Without duplicates.
- Multiset (bag)
  - Set-like container that allows duplicates.
- Multimap
  - Similar to map. Associates values with keys.
  - In multimap, same key can be mapped to multiple values.

# Set

- add(e)
  - Adds the element *e* to *S* (if not already present)
- remove(e)
  - Removes the element *e* from *S* (if present)
- contains(e)
  - Returns whether *e* is an element of *S*
- iterator()
  - Returns an iterator of elements in *S*.

# Summary

- We use **sets** if we do not want any duplicate values in a list.
    - We can turn a set to array by toArray. But for arrays we need to create a set and write them in it.
-

```java
import java.util.HashSet;
import java.util.Set;

public class Main {
    public static void main(String[] args) {
        Set<String> mySet = new HashSet<>();
        mySet.add("apple");
        mySet.add("banana");

        System.out.println(mySet);

        mySet.add("apple");
        System.out.println(mySet);
}
}
```

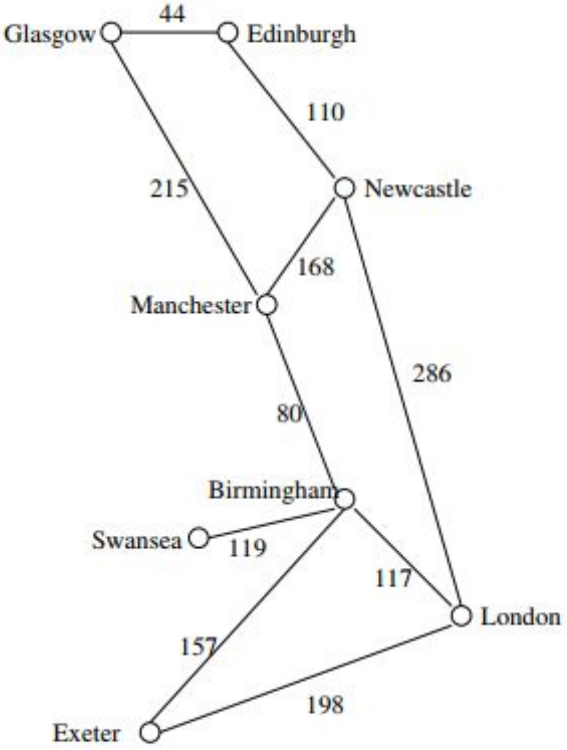
# Graphs

- Way of representing relationships that exists between pair of objects.
- A graph is a set of objects (**vertices**) together with a collection of pairwise connection between them (**edges**).
- Many application domains:
  - Mapping
  - Transportation
  - Computer Networks
  - Electrical Engineering
- A graph G is simply a set V of vertices and collection E of pairs of vertices from V, called edges.
  - Thus, a graph is a way of representing connections or relationships between pairs of objects from some set V.

# Graphs

- Edges can be either **directed** or **undirected**.
- An edge (u,v) is said to be **directed** from *u* to *v* if the pair (u,v) is ordered, with u preceding v.
    - u is *origin*
    - v is *destination*
- An edge (u,v) is said to be **undirected** if the pair (u,v) is not ordered.
- Undirected edges are sometimes denoted with set notation: {u,v}

- If all the edges in a graph are undirected, then we say the graph is an **undirected graph**
- A graph whose edges are all directed is called a **directed graph** or **digraph**.
- A graph with both edges (directed and undirected) is called a **mixed graph**.
- If there are labels on the graph, we call them **weighted**
- Graph on the right is **undirected** and **weighted**.



# Example

- A city map can be modeled as a graph.
- Vertices are intersections or dead ends
- Edges are stretches of streets without intersections.
- It has both undirected edges
  - Stretches of two-way streets
- And directed edges
  - Stretches to one-way streets
- A graph modeling a city map is a *mixed graph*.

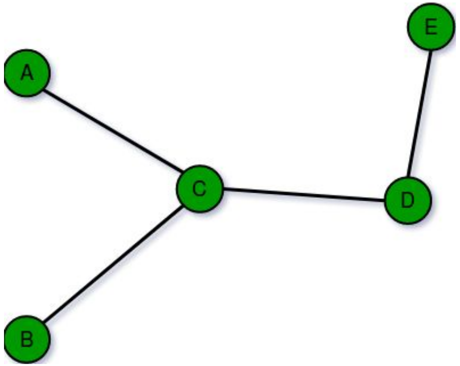
# Graphs

- These are known as graphs.
- Consists of a series of **nodes** (also called vertices or points), displayed as nodes.
- Edges (lines, links, arcs) displayed as *connections* between nodes.
- Lots of terminology.
    - A graph is said to be *simple* if it has no *self-loops*
        - Edges connected at both ends to the same vertex and no more than one edge connecting any pair of vertices.
        - We'll generally talk about simple graphs.
- If there are labels on the edges, we say that the graph is **weighted**.

# Types of Graphs

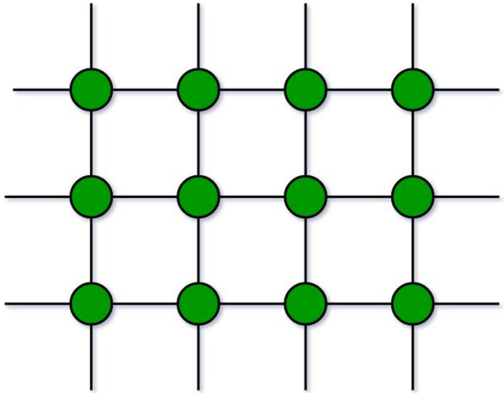
**Finite Graphs**

- Finite number of vertices and edges.
- They are limited and can be counted.
- Used to model real-world situations where there is a limited number of objects and relationships between nodes.




**Infinite Graphs**

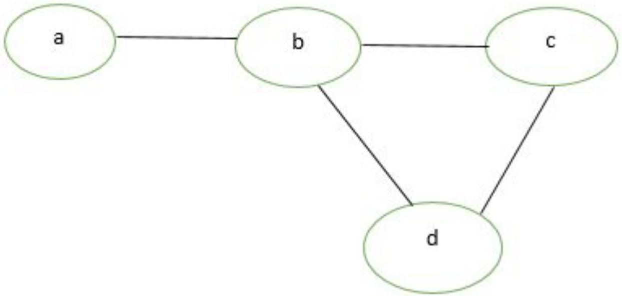
- Infinite number of vertices and edges.



# Types of Graphs

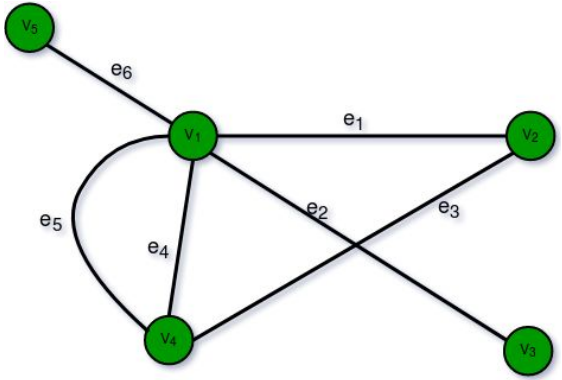
**Simple Graphs**

- Does not contain more than one edge between pair of vertices.
- e.g. A simple railway track connecting diff. cities




**Multi Graph**

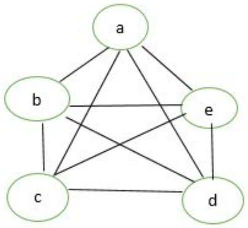
- Graph containing some parallel edges but doesn't contain any self-loop.
- e.g. Road Map
- Parallel Edges: e5, e4

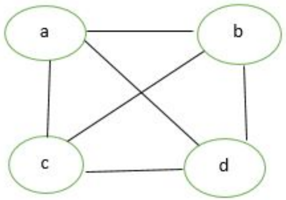


# Types of Graphs

**Complete Graph (Full Graph)**

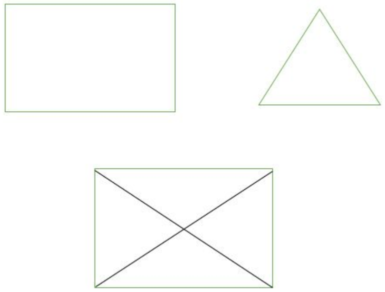
- A simple graph with n vertices is called a complete graph if the degree of each vertex is n-1
- One vertex is attached with n-1 edges.




**Regular Graph**

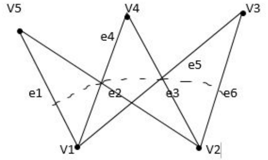
- A simple graph is said to be regular if all vertices of graph G are of equal degree.
- All complete graphs are regular but not all regular is not complete.
- A type of undirected graph where every vertex has the same number of neighbors.



# Types of Graphs

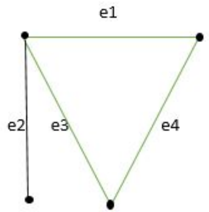
**Bipartite Graphs (Bigraph)**

- If vertex set V(G) can be partitioned into two non-empty disjoint subsets (U and V), where every edge connects a vertex in U to one in V.




**Labeled Graph (Weighted Graph)**

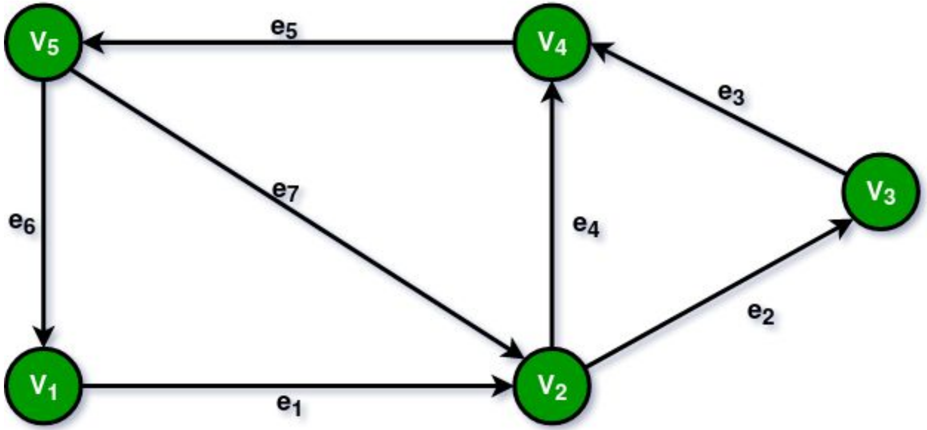
- Vertices and edges of a graph are labeled with name, date or weight.
- 



# Types of Graphs

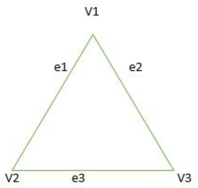
## Digraph Graph (Directed Graph)

- A graph with mapping f such that every edge maps onto some ordered pair of vertices.




## Cyclic Graph

- A graph that contains at least one graph cycle.
  - It is a path or sequence of edges and vertices wherein a vertex is reachable from itself through a sequence of edges.
  - Path starts and ends on the same vertex and contains to repeating vertices.

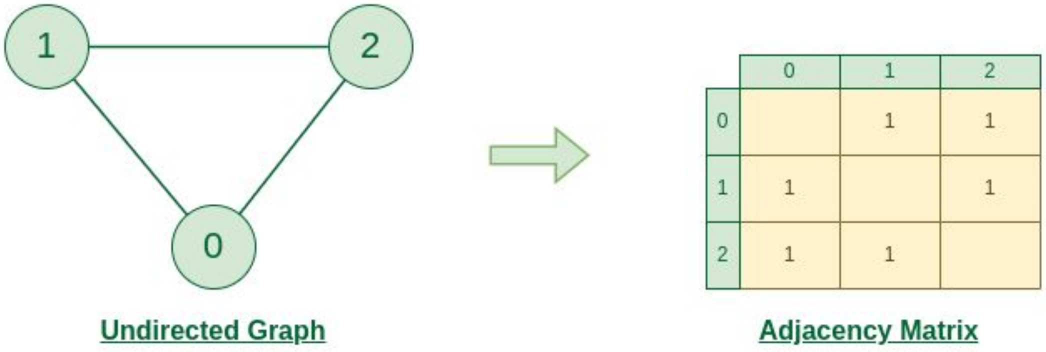


# Graph representation

- Two common ways
  - Adjacency Matrix
  - Adjacency List

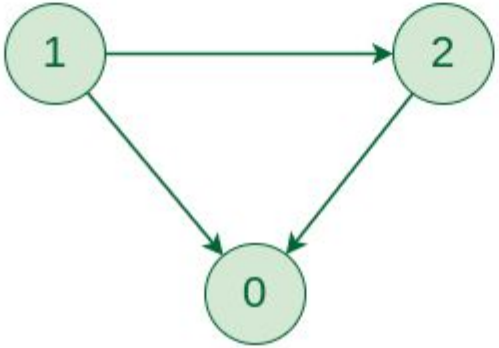
# Adjacency Matrix

- A square matrix to represent a finite graph.
- If the graph is undirected (all edges are bidirectional)
  - The adjacency matrix is symmetric.




Graph Representation of Undirected graph to Adjacency Matrix

# Representation of a directed graph




Directed Graph

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | | | |
| 1 | 1 | | 1 |
| 2 | 1 | | |

Adjacency Matrix

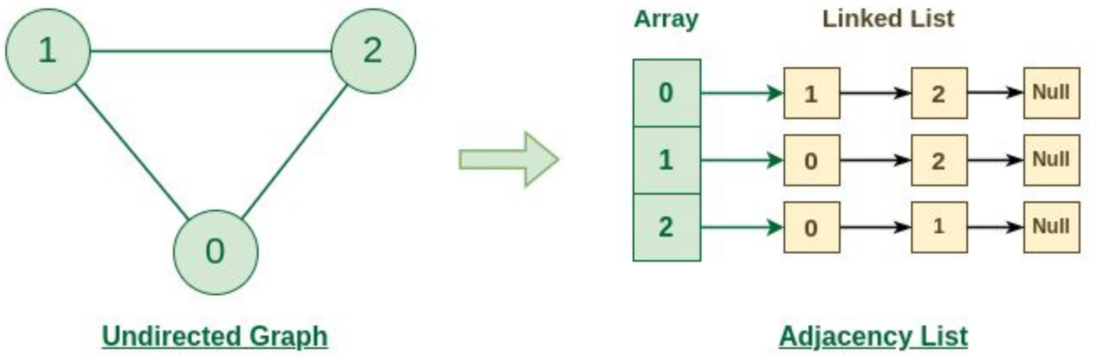
Graph Representation of Directed graph to Adjacency Matrix

# Adjacency List

- An array of Lists is used to store edges between two vertices.
- The size of array is equal to the number of vertices (i.e, n).
- Each index in this array represents a specific vertex in the graph.
- The entry at the index i of the array contains a *linked list* containing the vertices that are adjacent to vertex i.
    - adjList[0] will have all the nodes which are connected (neighbour) to vertex 0.
    - adjList[1] will have all the nodes which are connected (neighbour) to vertex 1 and so on.

# Adjacency List (undirected)

- 3 vertices
- We need an array of size 3.
- Each index will represent a vertex.
- Vertex 0 has two neighbors.
    - 1 and 2
    - So add them to index 0
- Vertex 1 has two: 0, 2
    - We add them
- Vertex has two: 0,1
    - We add them




Graph Representation of Undirected graph to Adjacency List

# Adjacency List (directed)

- 3 vertices
- Array of size 3.
- Vertex 0 has no neighbors.
  - Empty
- Vertex 1 has 2:
  - 0 and 2
  - We add them
- Vertex 2 has 1
  - Only 0
  - We add it.