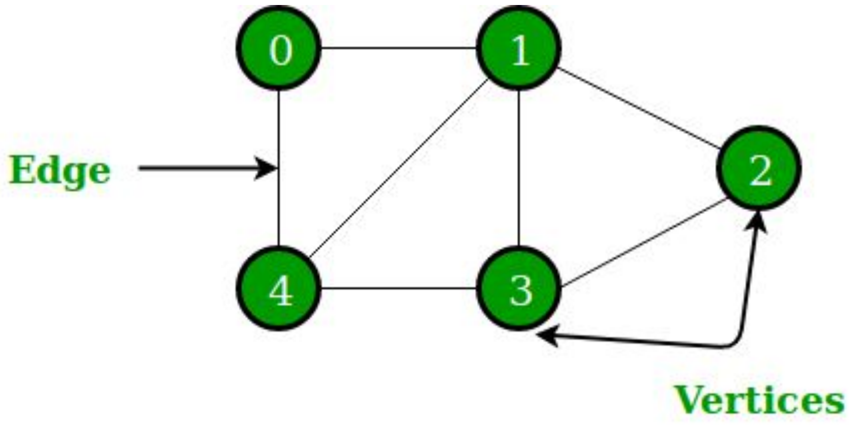
# Graphs

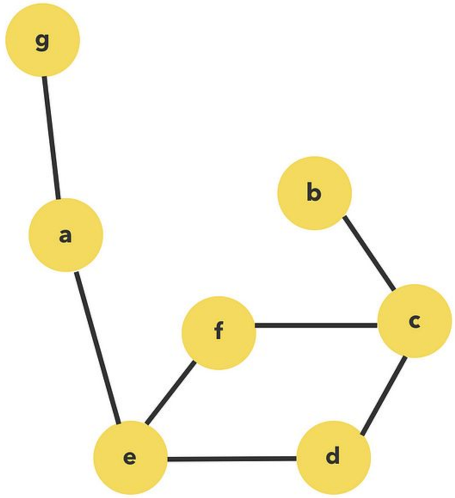
Fall 2024

# Graphs

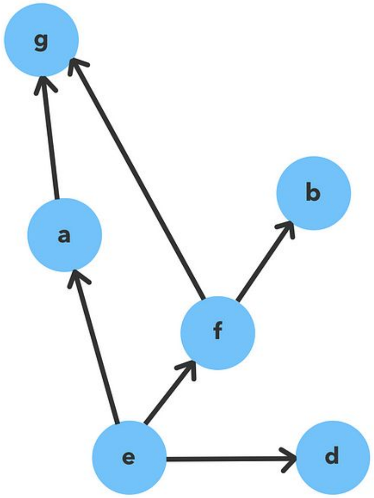
- A graph G is a pair (V, E), where V is a set of vertices (nodes) and E is a set of edges, which are pairs of vertices.
- Types
  - Directed
    - Edges have a direction, indicating a one-way relationship between vertices.
  - Undirected
    - Edges have no direction.
  - Weighted
    - Edges have associated weights, representing a cost or distance.


0
1
2
3
4
Edge
Vertices

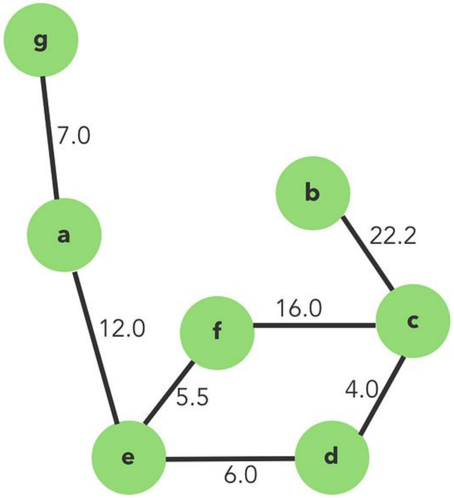
## Undirected




## Directed




## Weighted



# Graph representation

- Adjacency Matrix
  - A 2D matrix where the element at row i and column j represents the weight of the edge between vertices i and j.
  - Simple to implement, efficient for dense graphs.
  - Space-inefficient for sparse graphs.
- Adjacency List
  - An array of lists, where each list stores the neighbors of a vertex and their corresponding edge weights.
  - Space-efficient for sparse graphs.
  - Less efficient for operations like checking if an edge exists.

# Graph Traversal

- Breadth First
  - Visits all vertices at the same level before moving to the next level.
  - Finding shortest paths in unweighted graphs, network broadcasting.
- Depth-First Search
  - Explores as deep as possible along a branch before backtracking.
  - Topological sorting, cycle detection, finding connected components.

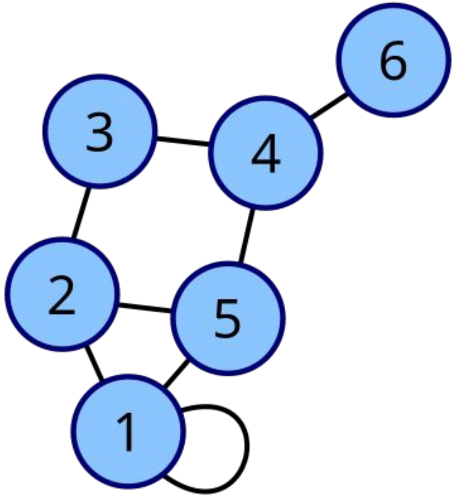
# Graph Algorithms

- Minimum Spanning Tree
  - A subset of edges that connects all vertices with minimum total weight.
  - **Prim's Algorithm**
    - Greedy algorithm that starts with a single vertex and adds edges with minimum weight.
  - **Kruskal's Algorithm**
    - Greedy algorithm that sorts edges by weight and adds edges without forming cycles.
- Shortest Path Algorithm
  - Dijkstra's Algorithm
    - Finds the shortest path from a source vertex to all other vertices in a weighted graph.
  - Bellman-Ford Algorithm
    - Handles negative edge weights and detects negative cycles.
- Topological Sorting
  - Linear ordering of vertices such that for every directed edge uv, vertex u comes before v.
  - Task scheduling, dependency resolution.

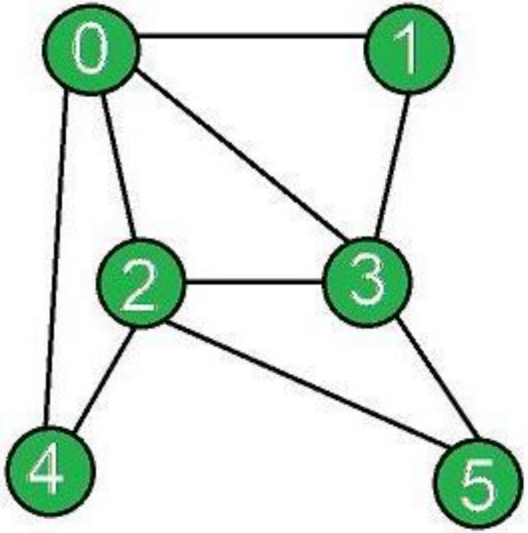
# Adjacency Matrix (connection matrix)

- Used to **represent** a finite graph.
- It is a **square** matrix.

# Adjacency matrix (undirected)

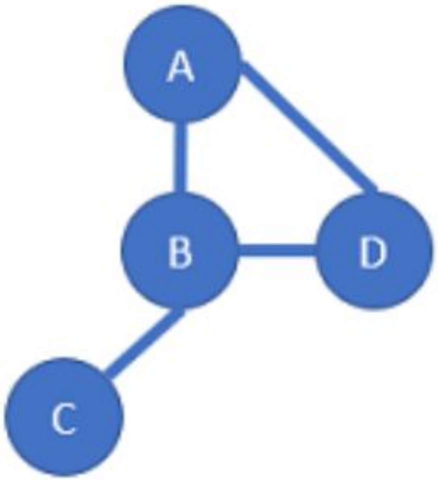



$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$




| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 2 | 1 | 0 | 0 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 0 | 0 | 1 |
| 4 | 1 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 0 | 1 | 1 | 0 | 0 |

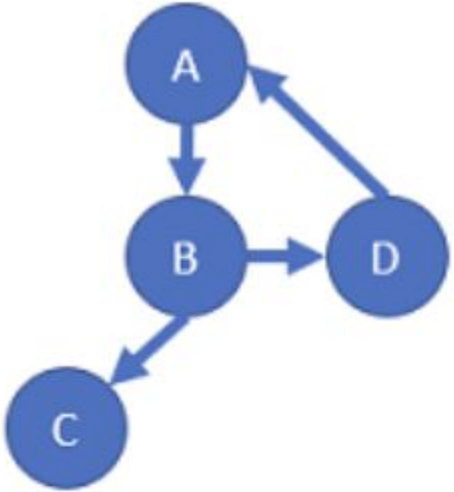
# Adjacency Matrix (Undirected)




$$
\begin{array}{c c}
 & \begin{array}{cccc} A & B & C & D \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \end{array} & \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}
\end{array}
$$

Undirected

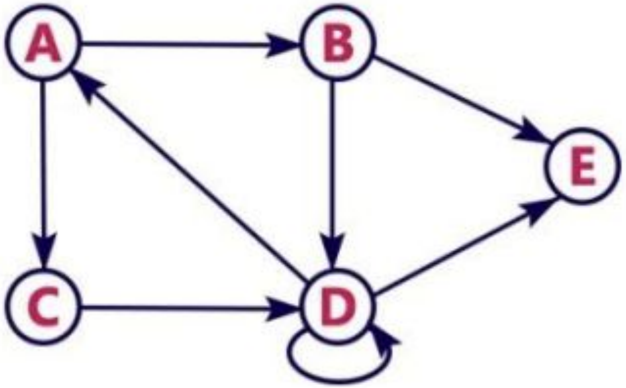
# Adjacency Matrix (Directed)



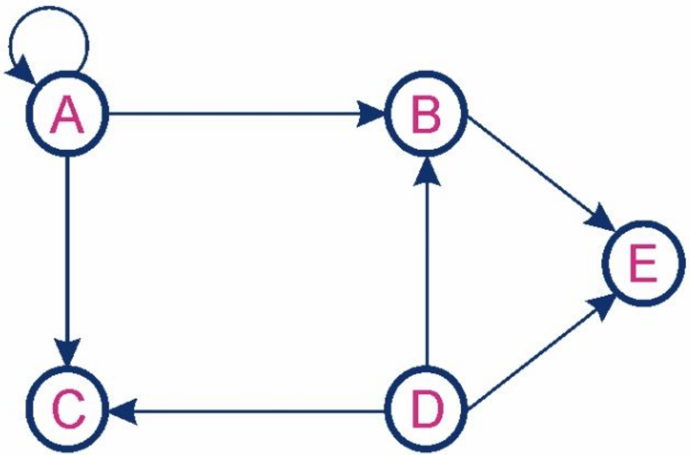

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Directed

# Directed



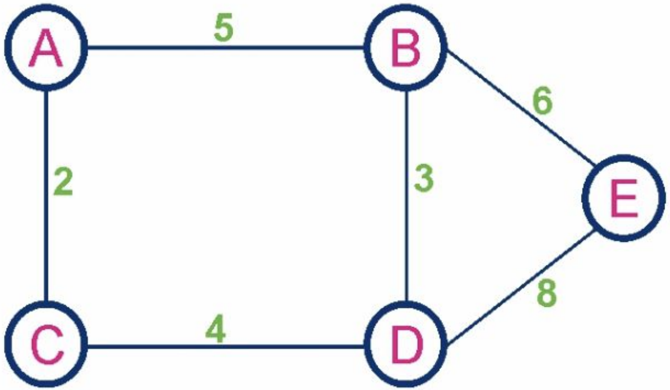

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$




Directed Graph

| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 1 | 1 | 1 | 0 | 0 |
| B | 0 | 0 | 0 | 0 | 1 |
| C | 0 | 0 | 0 | 0 | 0 |
| D | 0 | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |

# Adjacency Matrix (Weighted)




Weighted Graph

| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 5 | 2 | 0 | 0 |
| B | 5 | 0 | 0 | 3 | 6 |
| C | 2 | 0 | 0 | 4 | 0 |
| D | 0 | 3 | 4 | 0 | 8 |
| E | 0 | 6 | 0 | 8 | 0 |

# Adjacency List

- A collection of unordered lists used to represent a finite graph.
    - Each unordered list with an *adjacency list* describes the set of neighbors of a particular **vertex** in a graph.
- 

# Traversals

- Breadth-First
  - Queue
- Depth-First
  - Stack