

# VE281

## Data Structures and Algorithms

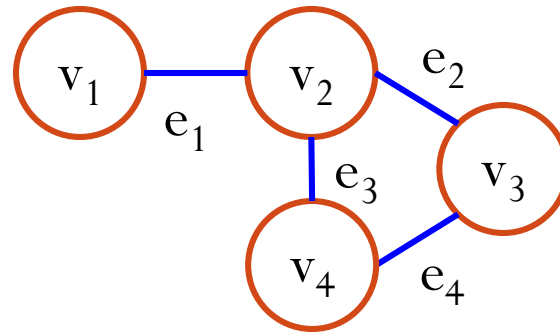
### **Graphs**

#### **Learning Objectives:**

- Know some basics about graph
- Know how to represent graphs in computer

# Graphs

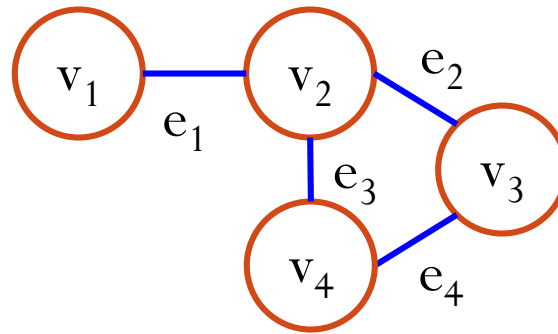
- A **graph** is a set of **nodes**  $V = \{v_1, v_2, \dots, v_n\}$  and **edges**  $E = \{e_1, e_2, \dots, e_m\}$  that connects pairs of nodes.
  - Nodes also known as **vertices**.
  - Edges also known as **arcs**.



- Two nodes are **directly connected** if there is an edge connecting them, e.g.,  $v_1$  and  $v_2$  are directly connected, but not  $v_1$  and  $v_3$ .

# Graphs

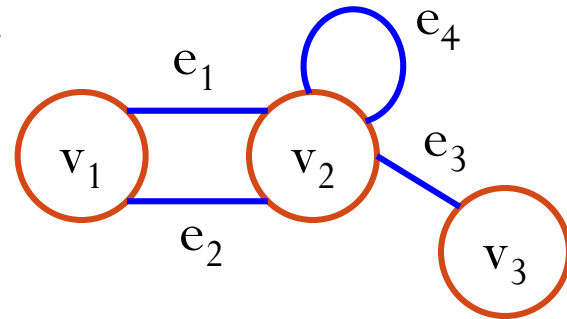
- Directly connected nodes are **adjacent** to each other (e.g.,  $v_1$  and  $v_2$ ), and one is the **neighbor** of the other.



- The edge directly connecting two nodes are **incident** to the nodes, and the nodes **incident** to the edge.

# Simple Graphs

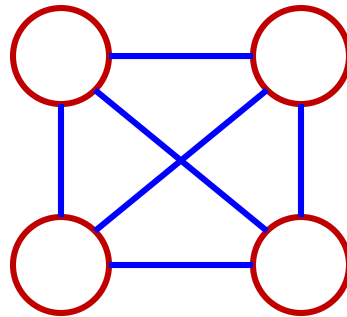
- Two nodes may be directly connected by more than one **parallel edges**, e.g.,  $e_1$  and  $e_2$ .



- An edge connecting a node to itself is called a **self-loop**, e.g.,  $e_4$ .
- A **simple graph** is a graph without parallel edges and self-loops.
  - Unless otherwise specified, we will work only with simple graphs in this course.

# Complete Graphs

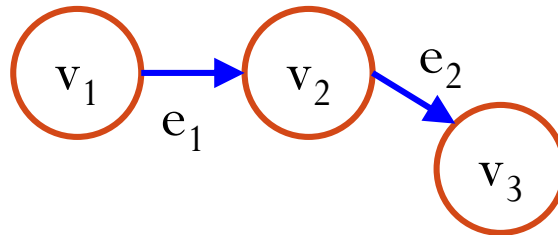
- A **complete graph** is a graph where every pair of nodes is directly connected.



- How many edges are there in a complete graph of  $N$  nodes?

# Directed Graphs

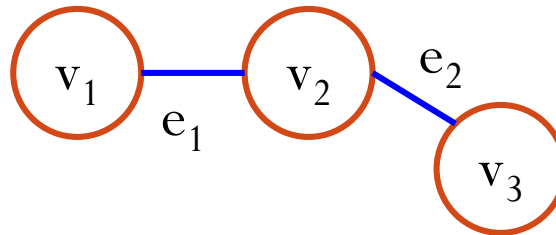
- **Directed graph** (digraph): edges are directional.



- Nodes incident to an edge form an **ordered** pair.
  - $e = (v_1, v_2)$  means there is an edge **from**  $v_1$  **to**  $v_2$ . However, there is no edge **from**  $v_2$  **to**  $v_1$ .
- Examples: rivers and streams, one-way streets, provider-customer relationships.

# Undirected Graphs

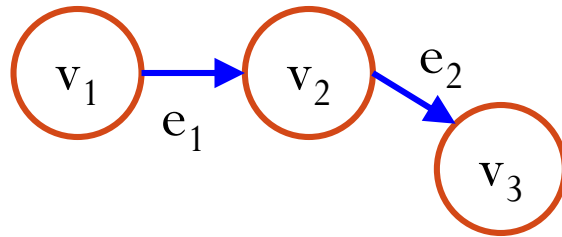
- **Undirected graph**: all edges have no orientation.



- There is no ordering of nodes on edges.
  - $e = (v_1, v_2)$  means there is an edge **between**  $v_1$  **and**  $v_2$ .
- Examples: friendship and two-way roads.

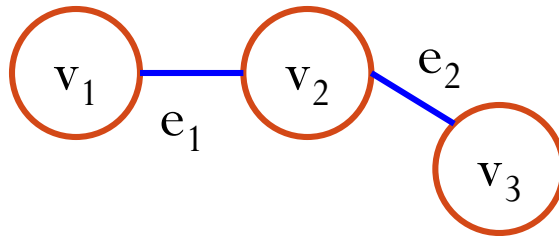
# Paths

- A **path** is a series of nodes  $v_1, \dots, v_n$  that are connected by edges.
- For a directed graph, if  $v_1, \dots, v_n$  is a path, then there is an edge **from**  $v_i$  **to**  $v_{i+1}$  for each  $i$ .



$v_1, v_2, v_3$  is a path.  
 $v_3, v_2, v_1$  is **not** a path.

- For an undirected graph, if  $v_1, \dots, v_n$  is a path, then there is an edge **between**  $v_i$  **and**  $v_{i+1}$  for each  $i$ .

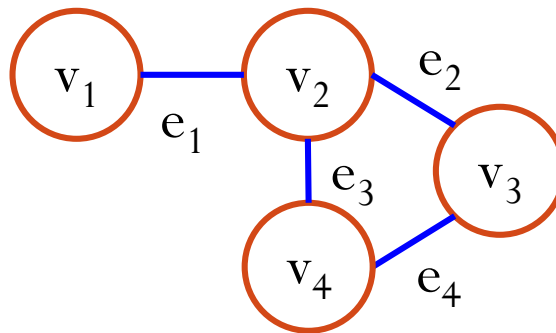


$v_1, v_2, v_3$  is a path.  
 $v_3, v_2, v_1$  is **also** a path.



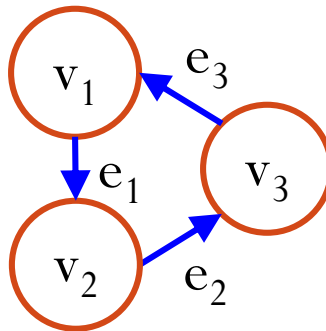
# Simple Paths

- A **simple path** is a path with no node appearing twice
  - e.g.,  $v_1, v_2, v_3$  is a simple path;  $v_1, v_2, v_3, v_4, v_2$  is not.



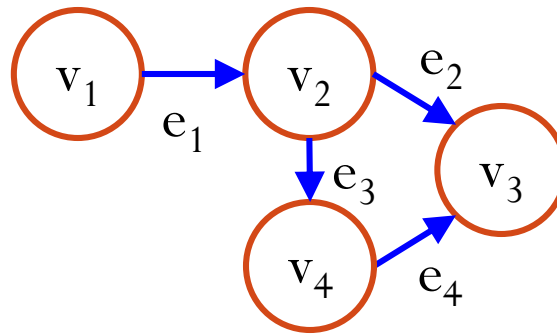
# Connected Graphs

- A **connected graph** is a graph where a simple path exists between all pairs of nodes.
- A directed graph is **strongly connected** if there is a simple **directed path** between any pair of nodes.



# Connected Graphs

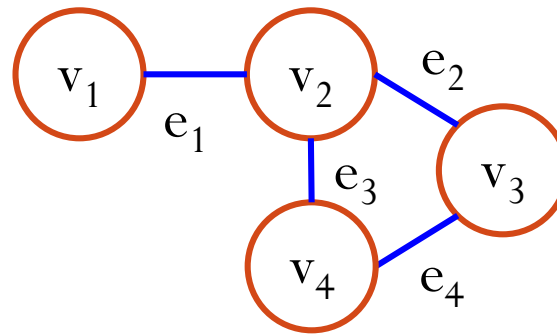
- A directed graph is **weakly connected** if there is a simple path between any pair of nodes in the underlying undirected graph.



The directed graph is weakly connected, but not strongly connected.

# Node Degree

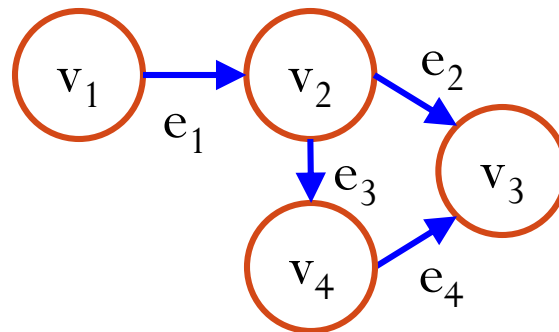
- The **degree** of a node is the number of edges incident to the node, e.g.,  $\text{degree}(v_2) = 3$ ,  $\text{degree}(v_3) = 2$ .



- What is the relationship between the sum of degrees of all nodes and the number of edges?
  - $\text{Sum}(\text{degrees}) = 2 * \text{Number}(\text{edges})$

# Node Degree for Directed Graphs

- For directed graphs, we differentiate between **incoming** edges and **outgoing** edges of a node. Thus we differentiate between a node's **in-degree** and its **out-degree**.
  - in-degree: number of incoming edges of a node
  - out-degree: number of outgoing edges of a node

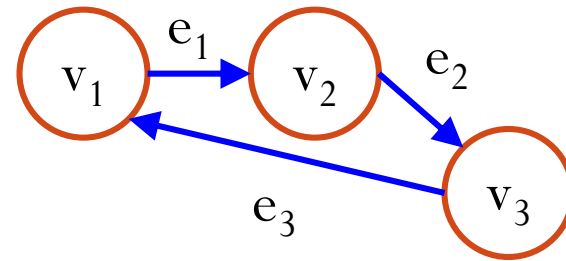


$$\begin{aligned}\text{in-degree}(v_2) &= 1 \\ \text{out-degree}(v_2) &= 2\end{aligned}$$

- Nodes with zero in-degree are **source** nodes, e.g.,  $v_1$ .
- Nodes with zero out-degree are **sink** nodes, e.g.,  $v_3$ .
- What is the sum of in-degrees/out-degrees of all nodes?

# Cycles and Directed Acyclic Graphs

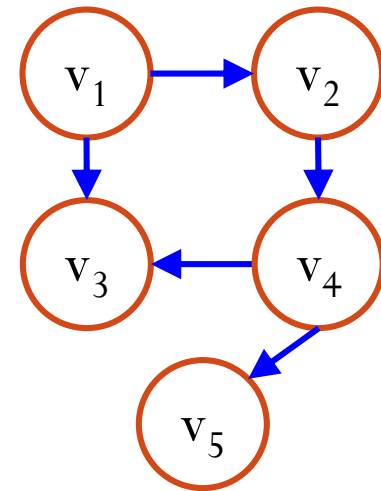
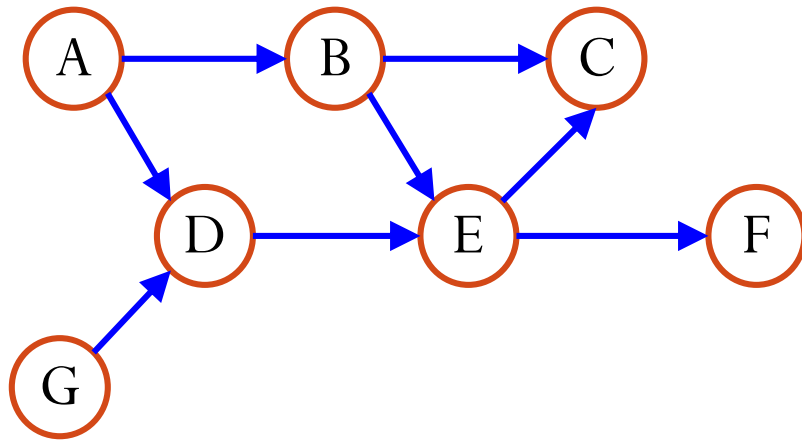
- A **cycle** is a path starting and finishing at the same node.
  - A self-loop is a cycle of length 1.
  - A **simple cycle** has no repeated nodes, except the first and the last node, e.g.,  $v_1, v_2, v_3, v_1$ .



- A graph with **no cycle** is called an **acyclic graph**.
- A **directed** graph **with no cycles** is called a **directed acyclic graph**, or **DAG** for short.

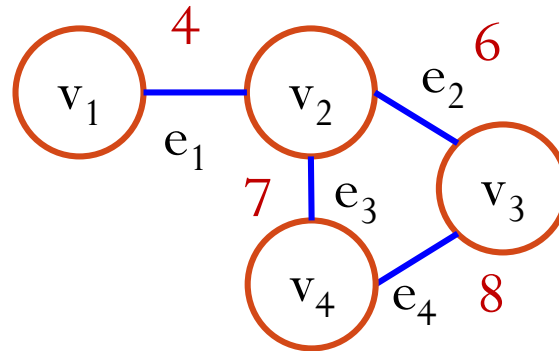
# Directed Acyclic Graphs (DAG)

- Are the following graphs DAGs?



# Weighted Graphs

- Weighted graph: edges of a graph may have different costs or weights.
- For example, the weights on edges represent the distance between two nodes.

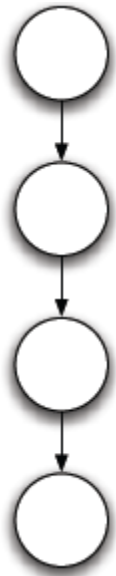




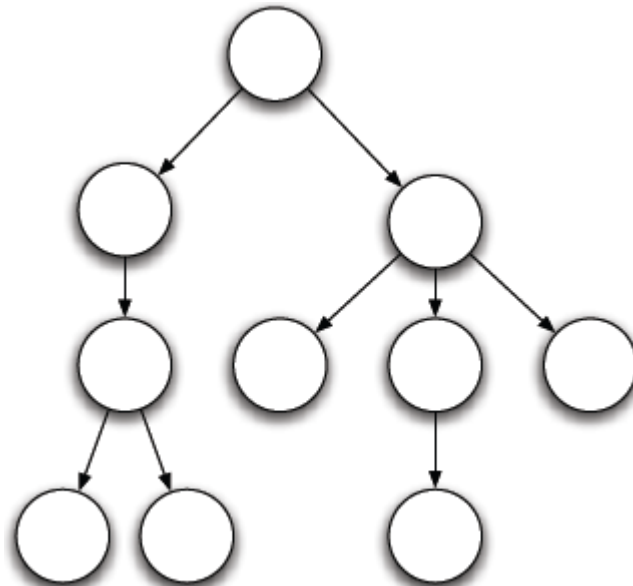
# Graph Size and Complexity

- The size of a graph and the complexity of a graph algorithms are usually defined in terms of
  - number of edges  $|E|$
  - number of vertices  $|V|$
  - or both
- **Sparse** graph: a graph with few edges.
  - $|E| \ll |V|^2$  or  $|E| \approx \Theta(|V|)$
  - Example: tree
- **Dense** graph: a graph with many edges.
  - $|E| \approx \Theta(|V|^2)$
  - Example: complete graph

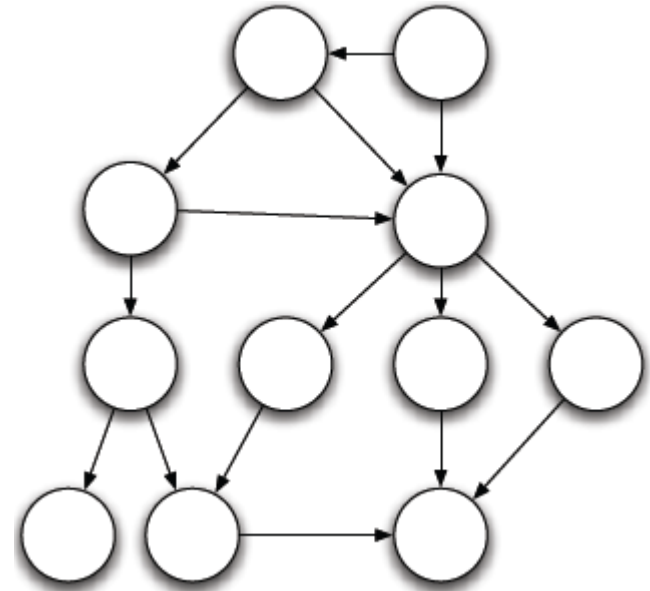
# Linked Lists, Trees, and Graphs



**Linked  
List**

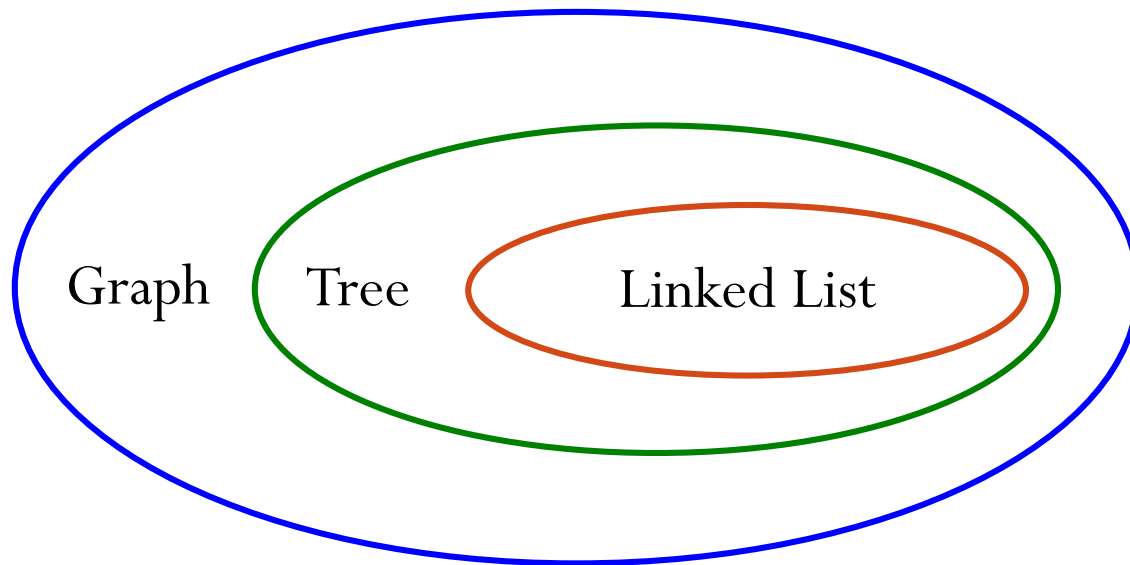


**Tree**



**Graph**

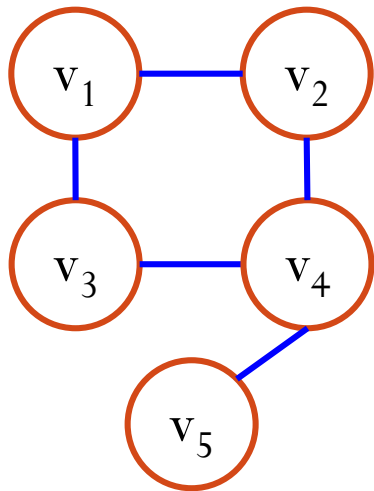
# Linked Lists, Trees, and Graphs



# Graph Representation

## Adjacency Matrix

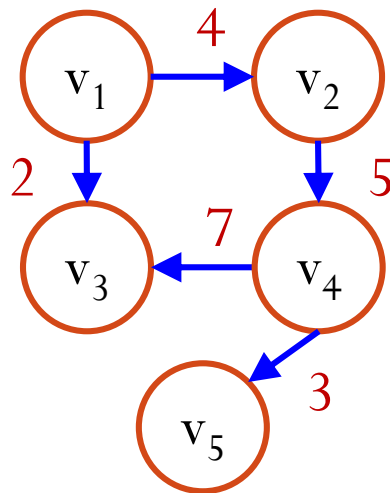
- Adjacency matrix: a  $|V| \times |V|$  matrix representation of a graph.
- $A(i, j) = 1$ , if  $(v_i, v_j)$  is an edge; otherwise  $A(i, j) = 0$ .



	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	0
3	1	0	0	1	0
4	0	1	1	0	1
5	0	0	0	1	0

# Adjacency Matrix for Weighted Graph

- If  $(v_i, v_j)$  is an edge and its weight is  $w_{ij}$ , then  $A(i, j) = w_{ij}$ ; otherwise  $A(i, j) = \infty$ .



	1	2	3	4	5
1	$\infty$	4	2	$\infty$	$\infty$
2	$\infty$	$\infty$	$\infty$	5	$\infty$
3	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
4	$\infty$	$\infty$	7	$\infty$	3
5	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$

Question: why not use 0 to represent a missing edge?

# Adjacency Matrix

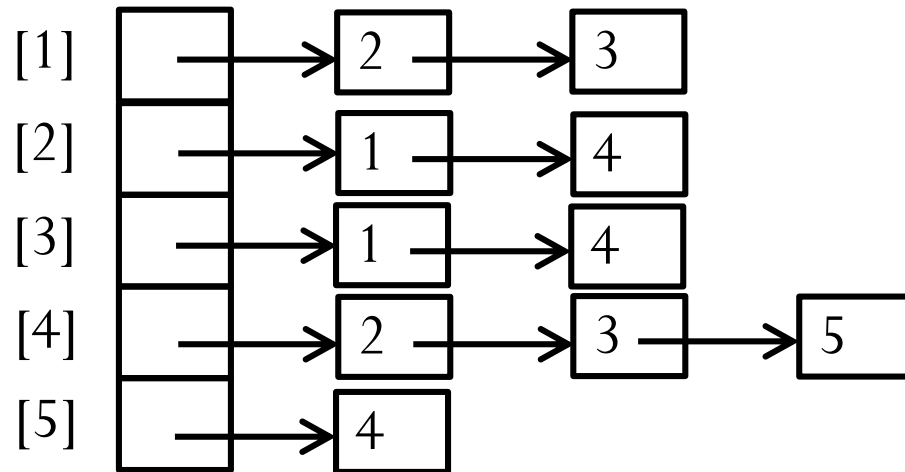
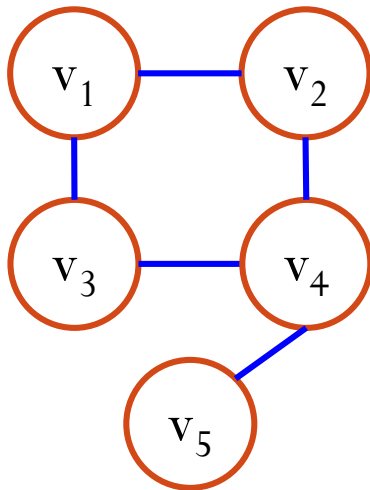
## Properties

- Space complexity:  $|V|^2$  units
  - For an undirected graph, may store only the lower or upper triangle. Thus,  $(|V| - 1)|V|/2$  units.
- What is the time complexity for finding if node  $v_i$  is adjacent to node  $v_j$ ?
  - $O(1)$
- What is the time complexity for finding **all** nodes adjacent to a given node  $v_i$ ?
  - $O(|V|)$

# Graph Representation

## Adjacency List

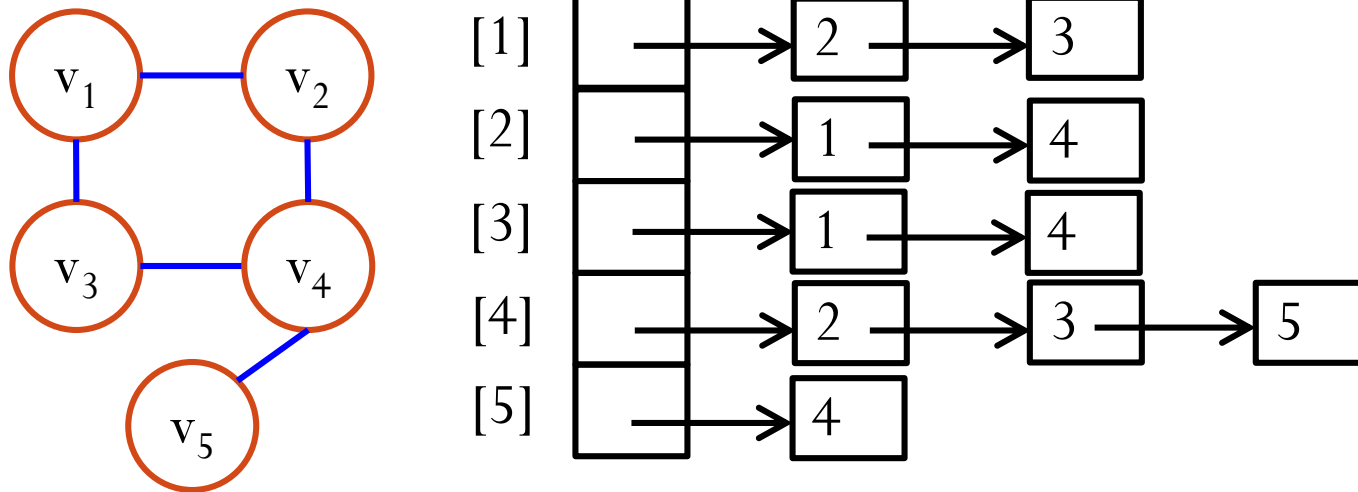
- Adjacency list: an array of  $|V|$  linked lists.
  - Each array element represents a node and its linked list represents the node's neighbors.



# Graph Representation

## Adjacency List

- Each edge in an undirected graph is represented twice.
  - Each edge is treated as **bidirectional**.

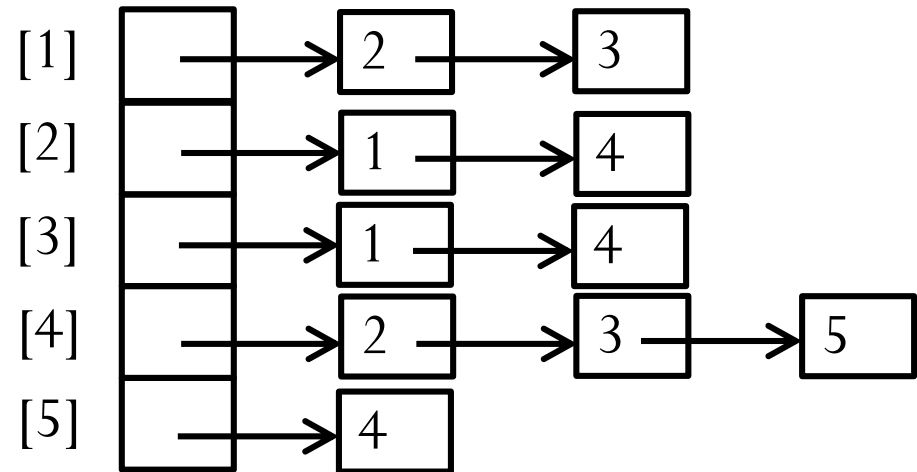
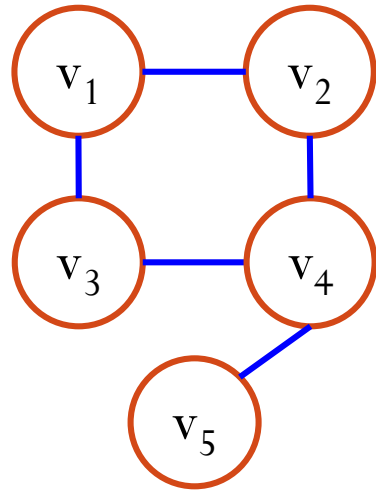


- Each edge in a directed graph is represented once.
- Weighted graph stores edge weight in linked-list node.



# Adjacency List

## Properties



- What is the space complexity?  $O(|E| + |V|)$
- What is the **worst case** time complexity for checking if node  $v_i$  is adjacent to node  $v_j$ ?  $O(|V|)$
- What is the **worst case** time complexity for finding all nodes adjacent to a given node  $v_i$ ?  $O(|V|)$

# Comparison of Graph Representation

- Worst case time complexity for two common operations:
  1. Determine whether  $v_i$  is adjacent to  $v_j$ 
    - Adjacency matrix:  $O(1)$ ; Adjacency list:  $O(|V|)$
  2. Determine all the nodes adjacent to  $v_i$ 
    - Both adjacency matrix and adjacency list:  $O(|V|)$
- Adjacency list often requires less space than adjacency matrix.
- Dense graphs are more efficiently represented as adjacency matrices and sparse graphs as adjacency lists.

# Sample Graph Problems

- Path finding problems
  - Find if there exists a path between two given nodes.
  - Find the shortest path between two given nodes.
- Connectedness problems
  - Find if the graph is a connected graph.