Graph theory

Introduction

We will define an Undirected Graph as a collection of *vertices*

$$V = \{v_1, v_2, ..., v_n\}$$

- The number of vertices is denoted by

$$|V| = n$$

- Associated with this is a collection E of <u>unordered</u> pairs $\{v_i, v_j\}$ termed *edges* which connect the vertices

There are a number of data structures that can be used to implement abstract undirected graphs

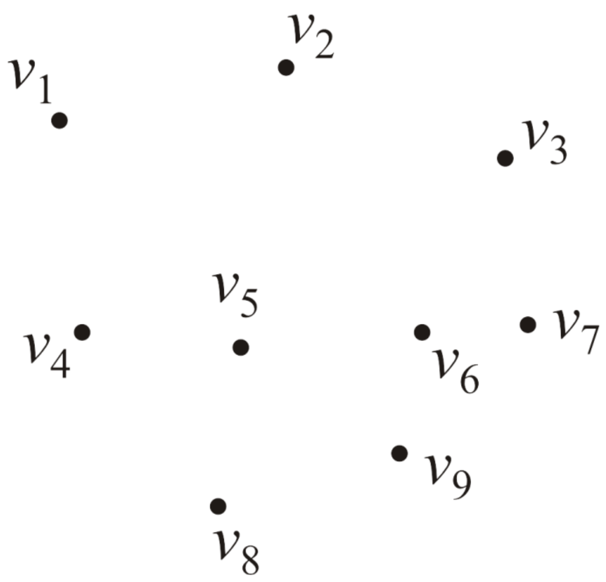
- Adjacency matrices
- Adjacency lists

Graphs

Consider this collection of vertices

$$V = \{v_1, v_2, \ldots, v_9\}$$

where $|V| = n$

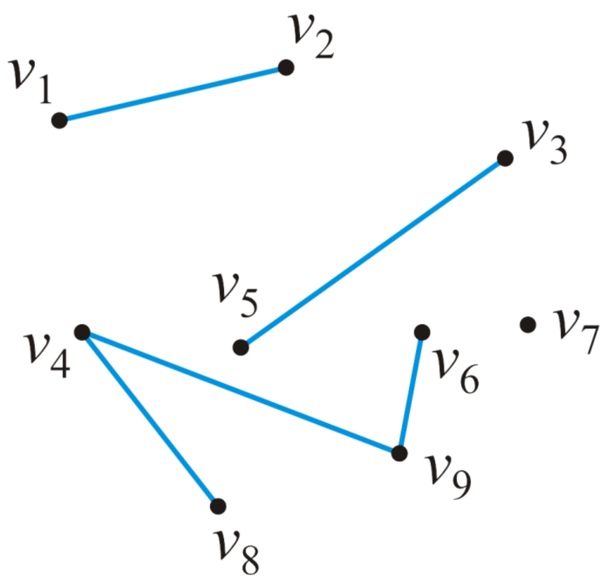


Undirected graphs

Associated with these vertices are $|E| = 5$ edges

$$E = \{\{v_1, v_2\}, \{v_3, v_5\}, \{v_4, v_8\}, \{v_4, v_9\}, \{v_6, v_9\}\}$$

- The pair $\{v_j , v_k\}$ indicates that both vertex v_j is adjacent to vertex v_k and vertex v_k is adjacent to vertex v_j



An undirected graph

Example: given the $|V| = 7$ vertices

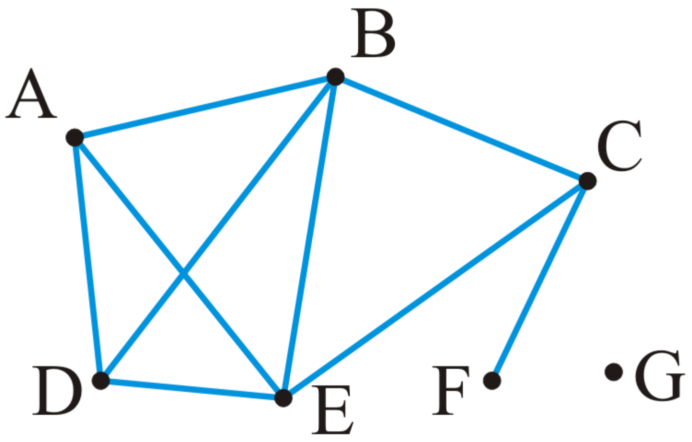
$$V = \{\mathrm{A, B, C, D, E, F, G}\}$$

and the $|E| = 9$ edges

$$E = \{\{\mathrm{A, B}\}, \{\mathrm{A, D}\}, \{\mathrm{A, E}\}, \{\mathrm{B, C}\}, \{\mathrm{B, D}\}, \{\mathrm{B, E}\}, \{\mathrm{C, E}\}, \{\mathrm{C, F}\}, \{\mathrm{D, E}\}\}$$

The maximum number of edges in an undirected graph is

$$|E| \le \binom{|V|}{2} = \frac{|V|(|V|-1)}{2} = \mathrm{O}\left(|V|^2\right)$$



Degree

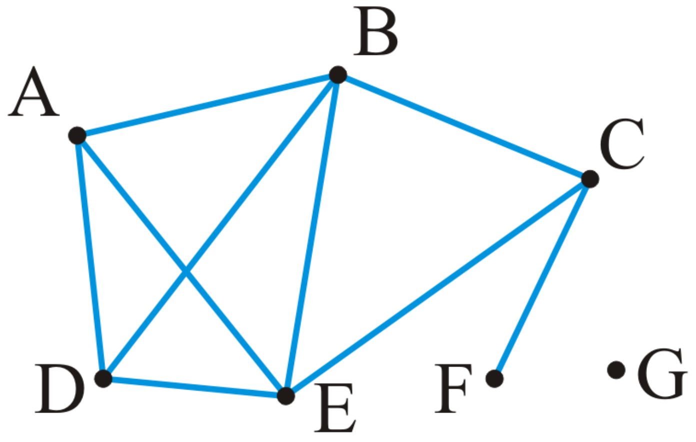
The degree of a vertex is defined as the number of adjacent vertices

degree(A) = degree(D) = degree(C) = 3

degree(B) = degree(E) = 4

degree(F) = 1

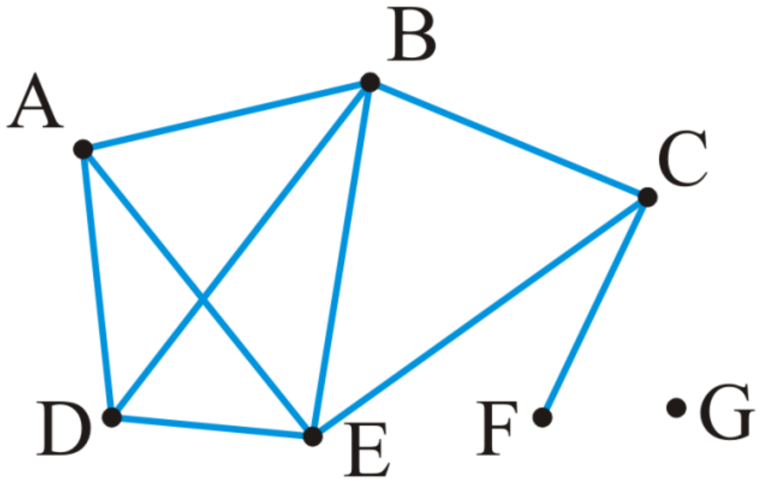
degree(G) = 0

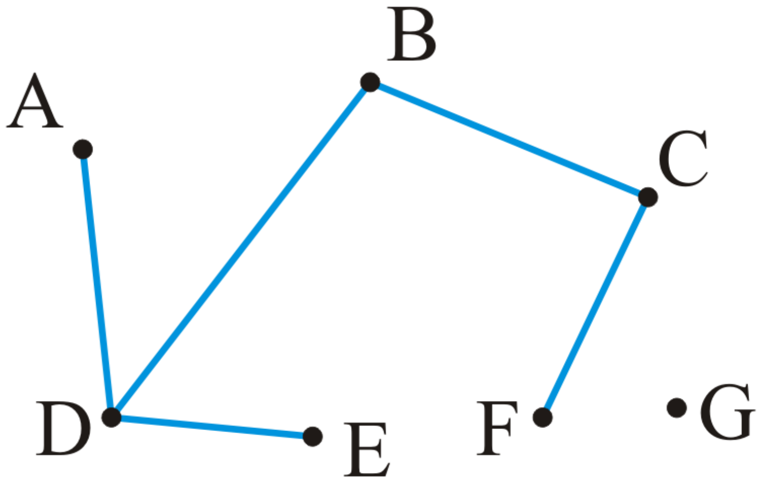


Those vertices adjacent to a given vertex are its *neighbors*

Sub-graphs

A *sub-graph* of a graph is a subset of the vertices and a subset of the edges





Paths

A path in an undirected graph is an ordered sequence of vertices

$$(v_0, v_1, v_2, \ldots, v_k)$$

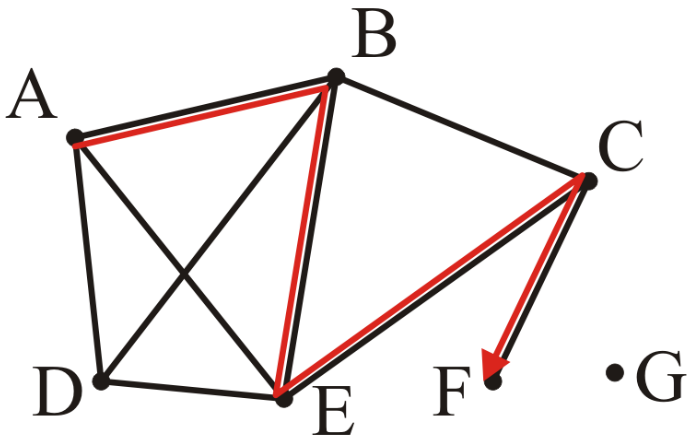
where $\{v_{j-1}, v_j\}$ is an edge for $j = 1, \ldots, k$

- Termed *a path from* v_0 to v_k
- The length of this path is k

A ***simple path*** has no repetitions other than perhaps the first and last vertices

A path of length 4:

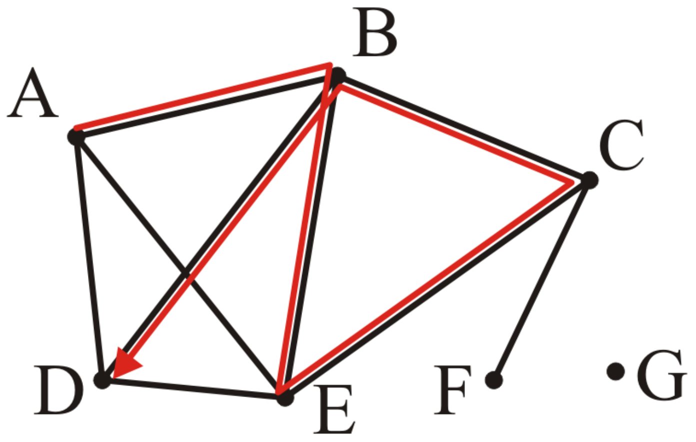
$$(\mathrm{A}, \mathrm{B}, \mathrm{E}, \mathrm{C}, \mathrm{F})$$



Paths

A path of length 5:

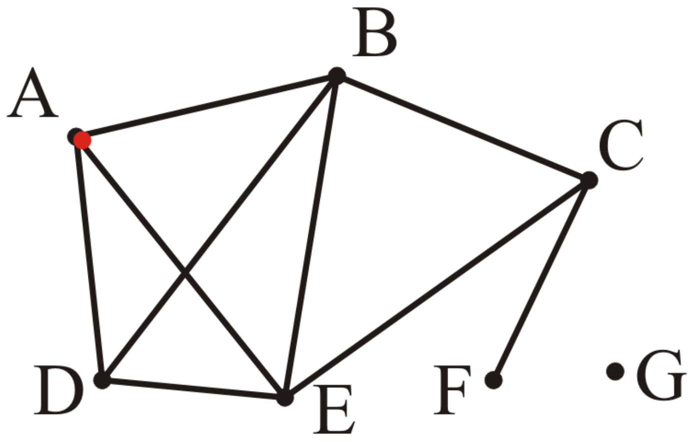
(A, B, E, C, B, D)



Paths

A *trivial* path of length 0:

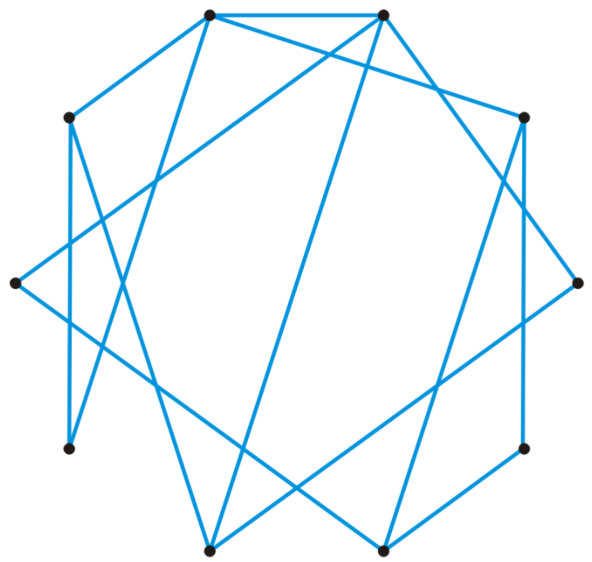
(A)



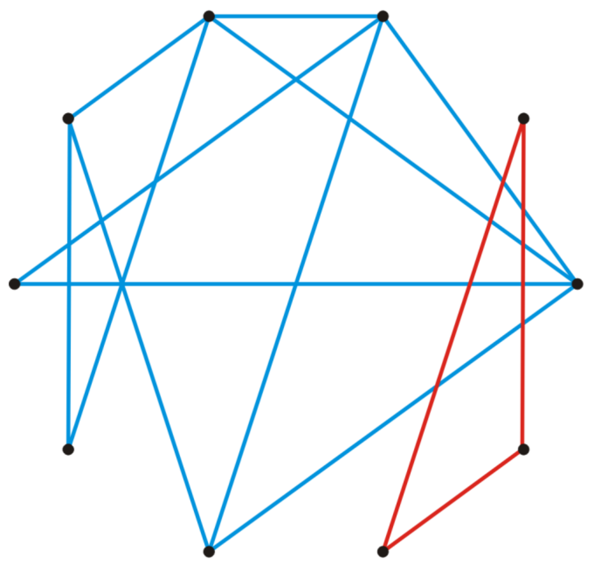
Connectedness

Two vertices v_i, v_j are said to be *connected* if there exists a path from v_i to v_j

A graph is connected if there exists a path between any two vertices



A connected graph

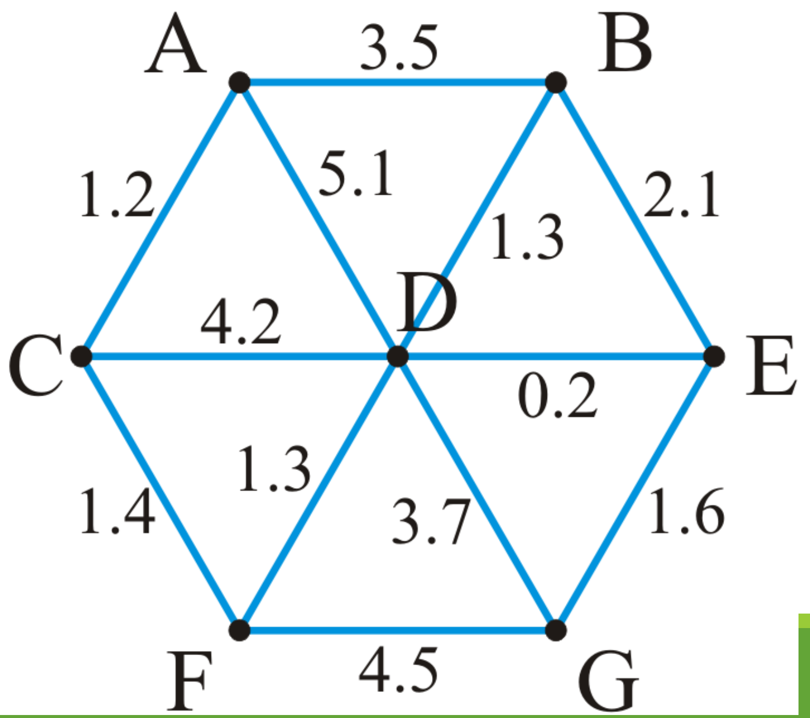


An unconnected graph

Weighted graphs

A weight may be associated with each edge in a graph
- This could represent distance, energy consumption, cost, etc.
- Such a graph is called a *weighted graph*

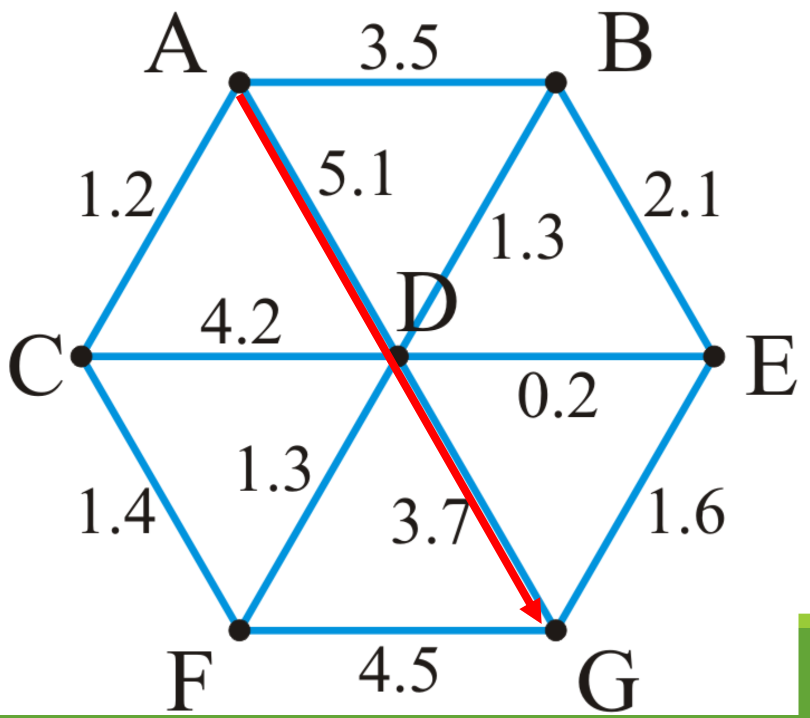
Pictorially, we will represent weights by numbers next to the edges



Weighted graphs

The *length* of a path within a weighted graph is the sum of all of the edges which make up the path

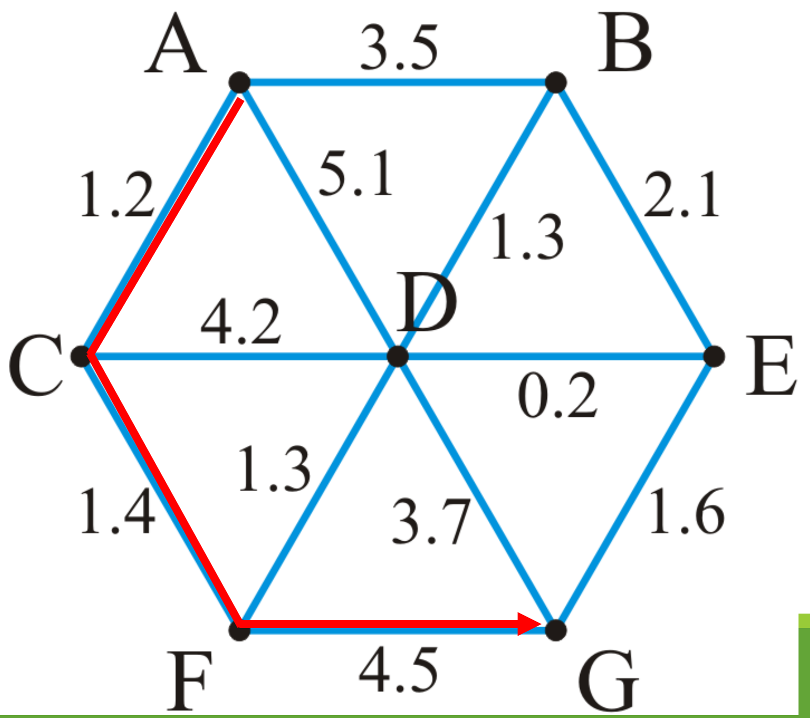
- The length of the path (A, D, G) in the following graph is $5.1 + 3.7 = 8.8$



Weighted graphs

Different paths may have different weights

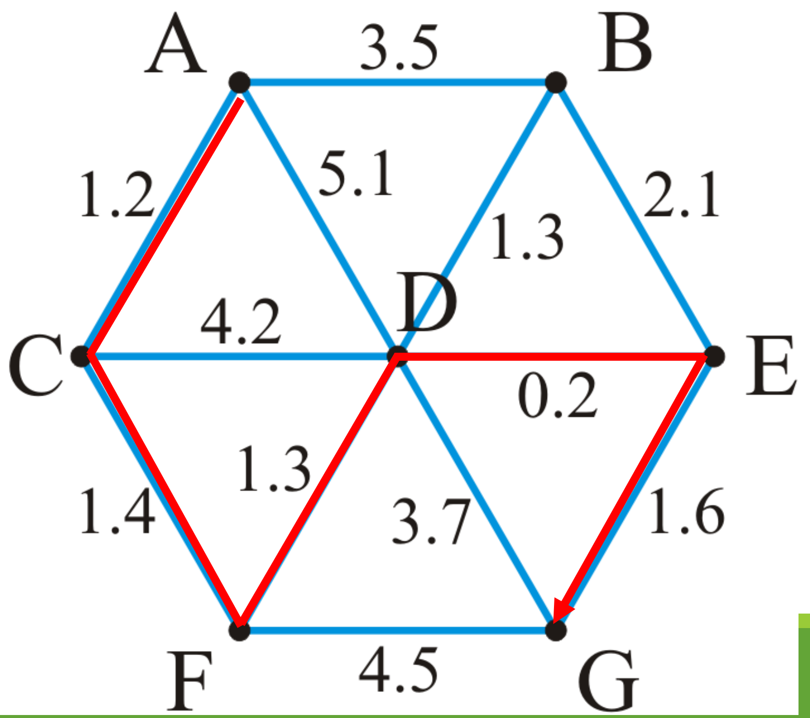
- Another path is (A, C, F, G) with length 1.2 + 1.4 + 4.5 = 7.1



Weighted graphs

Problem: find the shortest path between two vertices

Here, the shortest path from A to G is (A, C, F, D, E, G) with length 5.7



Directed graphs

In a *directed graph*, the edges on a graph are be associated with a direction

- Edges are ordered pairs (v_j, v_k) denoting a connection from v_j to v_k
- The edge (v_j, v_k) is different from the edge (v_k, v_j)

Streets are directed graphs:

- In most cases, you can go two ways unless it is a one-way street

The maximum number of directed edges in a directed graph is

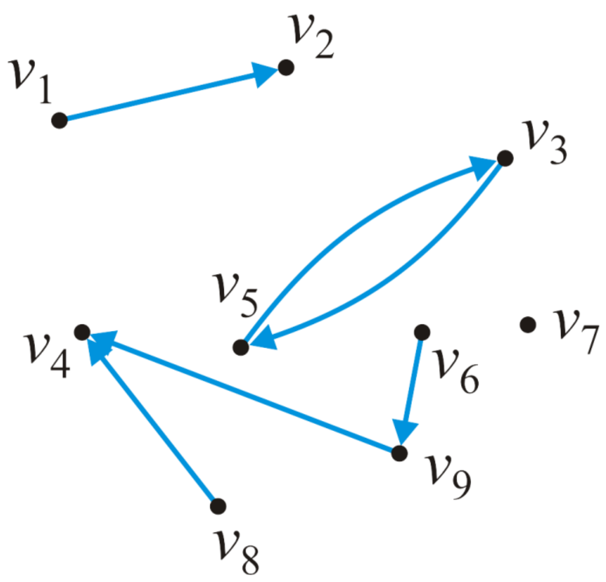
$$|E| \le 2\binom{|V|}{2} = 2\frac{|V|(|V|-1)}{2} = |V|(|V|-1) = \mathrm{O}\left(|V|^2\right)$$

Directed graphs

Given our graph of nine vertices $V = \{v_1, v_2, \ldots v_9\}$

- These six pairs (v_j, v_k) are *directed edges*

$$E = \{(v_1, v_2), (v_3, v_5), (v_5, v_3), (v_6, v_9), (v_8, v_4), (v_9, v_4)\}$$



In and out degrees

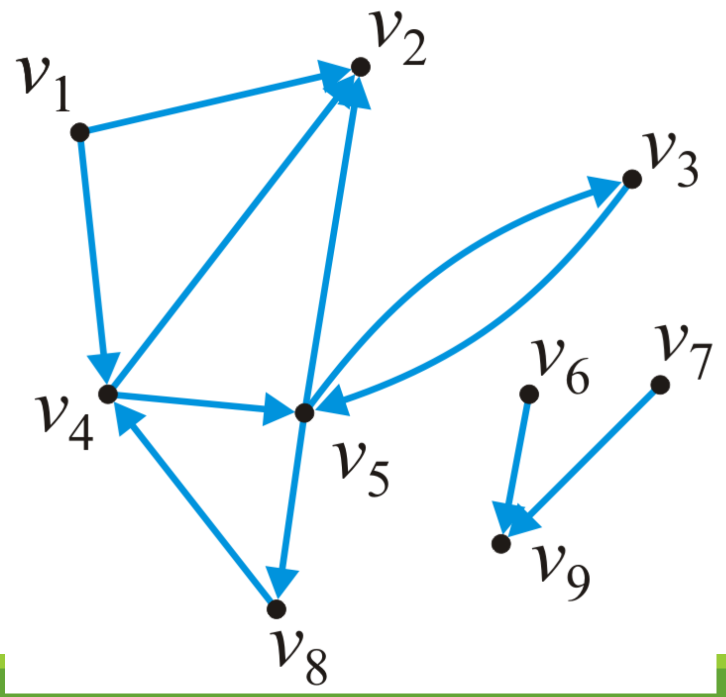
The degree of a vertex must be modified to consider both cases:

- The *out-degree* of a vertex is the number of vertices which are adjacent to the given vertex
- The *in-degree* of a vertex is the number of vertices which this vertex is adjacent to

In this graph:

in_degree(v_1) = 0 out_degree(v_1) = 2

in_degree(v_5) = 2 out_degree(v_5) = 3



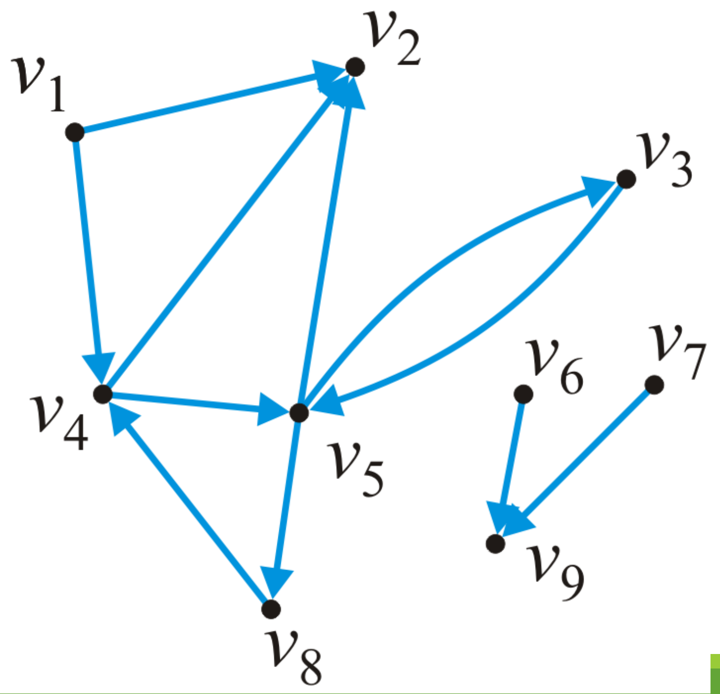
Sources and sinks

Some definitions:

- Vertices with an in-degree of zero are described as *sources*
- Vertices with an out-degree of zero are described as *sinks*

In this graph:

- Sources: v_1, v_6, v_7
- Sinks: v_2, v_9



Paths

A path in a directed graph is an ordered sequence of vertices

$$(v_0, v_1, v_2, \ldots, v_k)$$

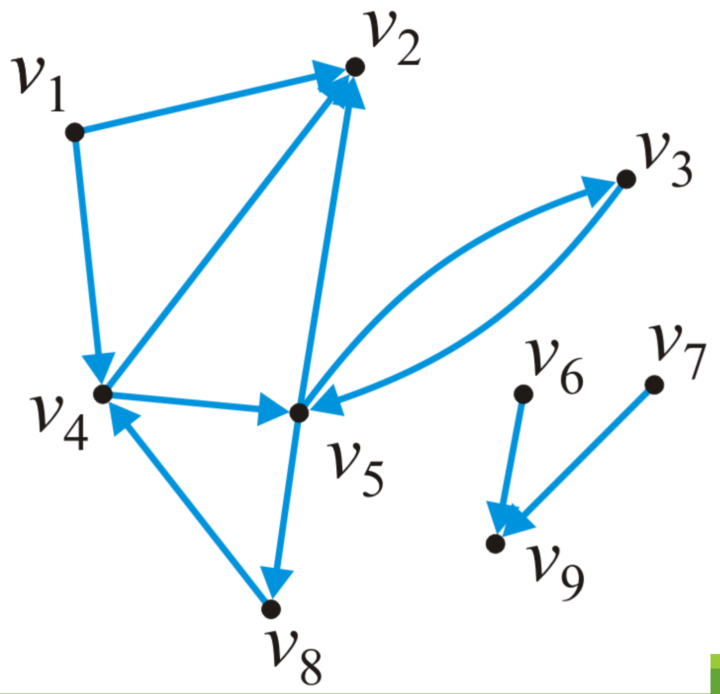
where (v_{j-1}, v_j) is an edge for $j = 1, \ldots, k$

A path of length 5 in this graph is

$$(v_1, v_4, v_5, v_3, v_5, v_2)$$

A simple cycle of length 3 is

$$(v_8, v_4, v_5, v_8)$$



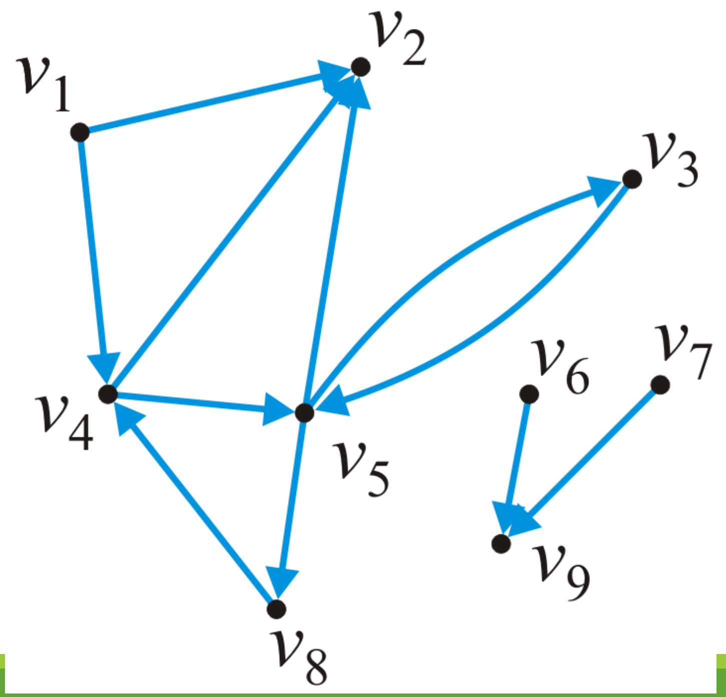
Connectedness

Two vertices v_j, v_k are said to be *connected* if there exists a path from v_j to v_k

- A graph is ***strongly connected*** if there exists a directed path between any two vertices
- A graph is ***weakly connected*** there exists a path between any two vertices that ignores the direction

In this graph:

- The sub-graph $\{v_3, v_4, v_5, v_8\}$ is strongly connected
- The sub-graph $\{v_1, v_2, v_3, v_4, v_5, v_8\}$ is weakly connected

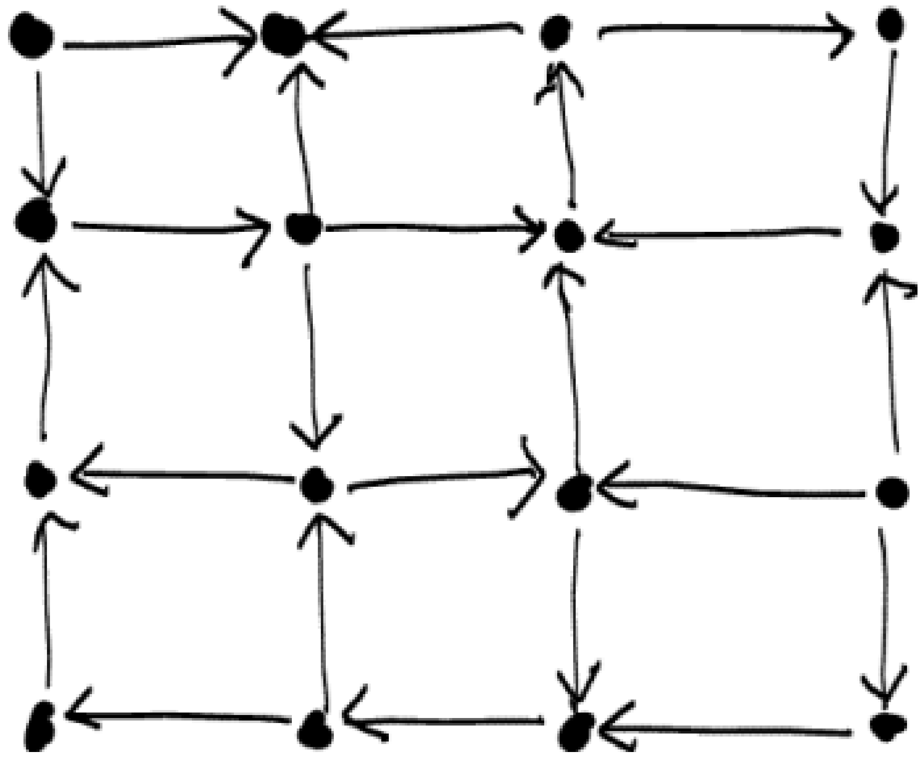


Strongly Connected Component

How many strongly connected components are available in this graph?

Who are they?

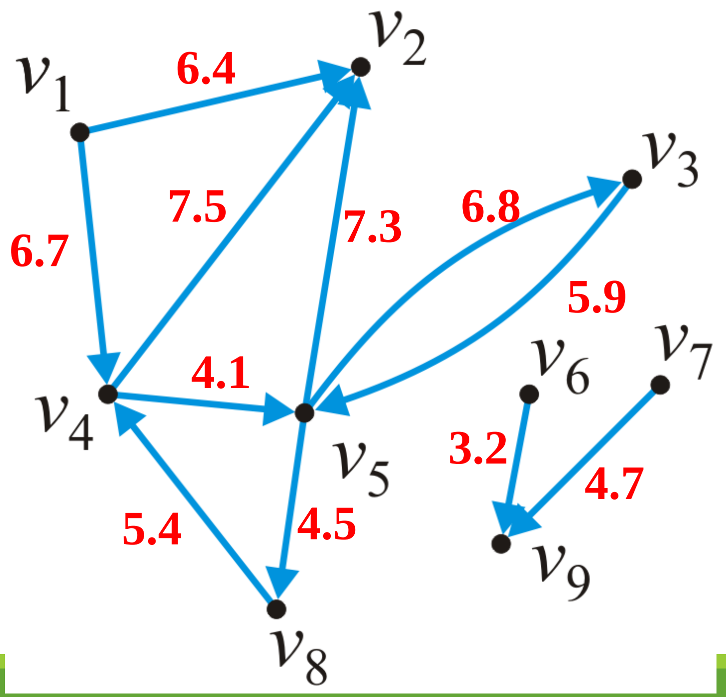
Bonus Mark Question



Weighted directed graphs

In a weighted directed graph, each edge is associated with a value

Unlike weighted undirected graphs, if both (v_j, v_k) and (v_j, v_k) are edges, it is not required that they have the same weight

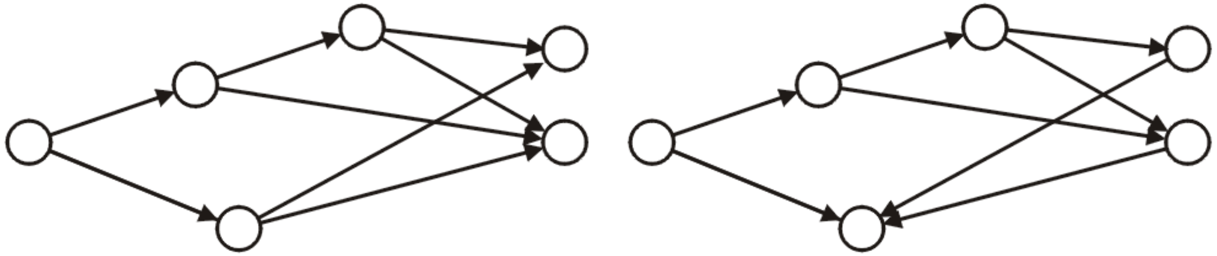


Directed acyclic graphs

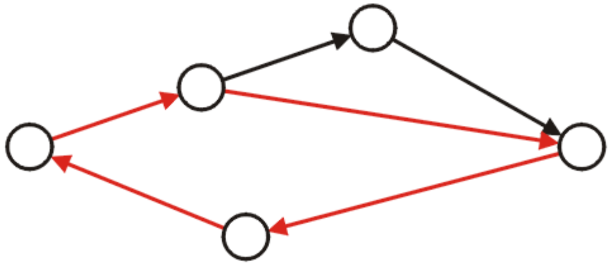
A *directed acyclic graph* is a directed graph which has no cycles

- These are commonly referred to as DAGs
- They are graphical representations of partial orders on a finite number of elements

These two are DAGs:



This directed graph is not acyclic:



Directed acyclic graphs

Applications of directed acyclic graphs include:

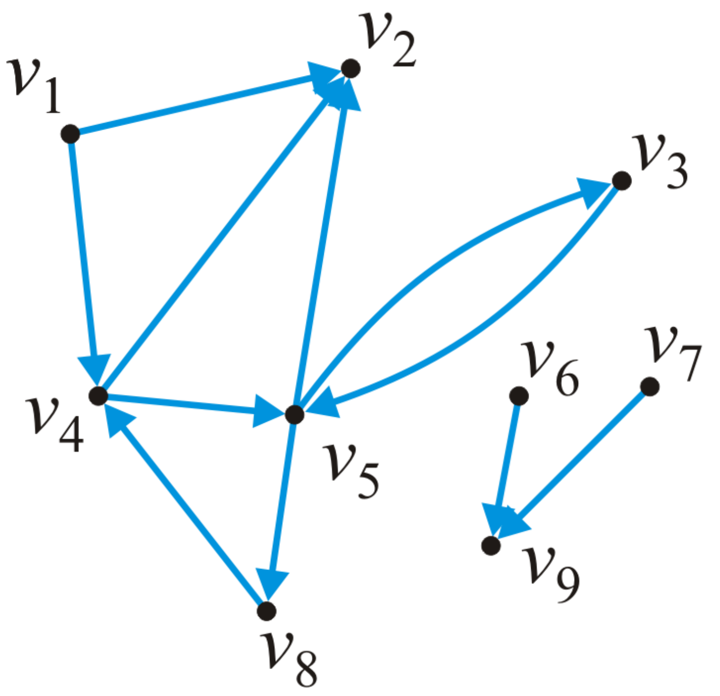
- The parse tree constructed by a compiler
- Dependency graphs such as those used in instruction scheduling and **`makefiles`**
- Dependency graphs between classes formed by inheritance relationships in object-oriented programming languages
- Information categorization systems, such as folders in a computer

Reference: http://en.wikipedia.org/wiki/Directed_acyclic_graph

Representations

How do we store the adjacency relations?

- Binary-relation list
- Adjacency matrix
- Adjacency list



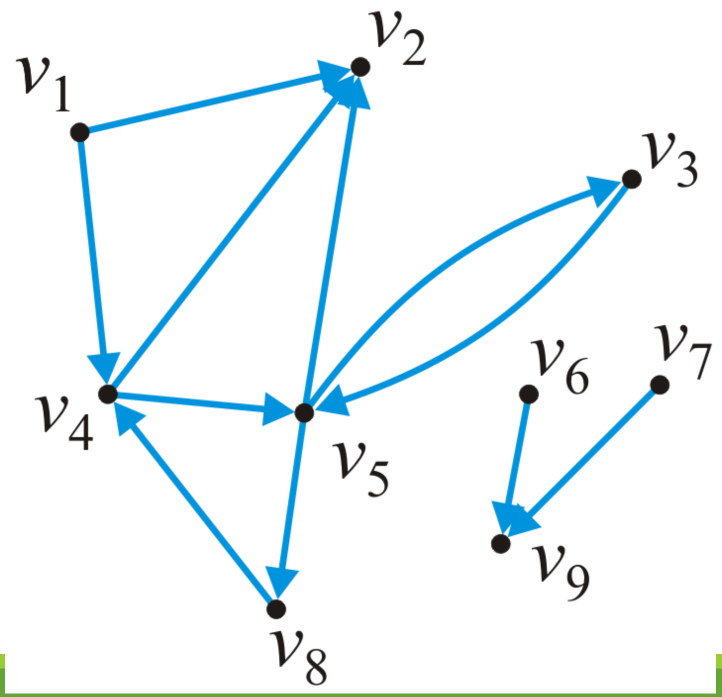
Binary-relation list

The most inefficient is a relation list:

- A container storing the edges

$$\{(1, 2), (1, 4), (3, 5), (4, 2), (4, 5), (5, 2), (5, 3), (5, 8), (6, 9), (7, 9), (8, 4)\}$$

- Requires $\Theta(|E|)$ memory
- Determining if v_j is adjacent to v_k is $O(|E|)$
- Finding all neighbors of v_j is $\Theta(|E|)$

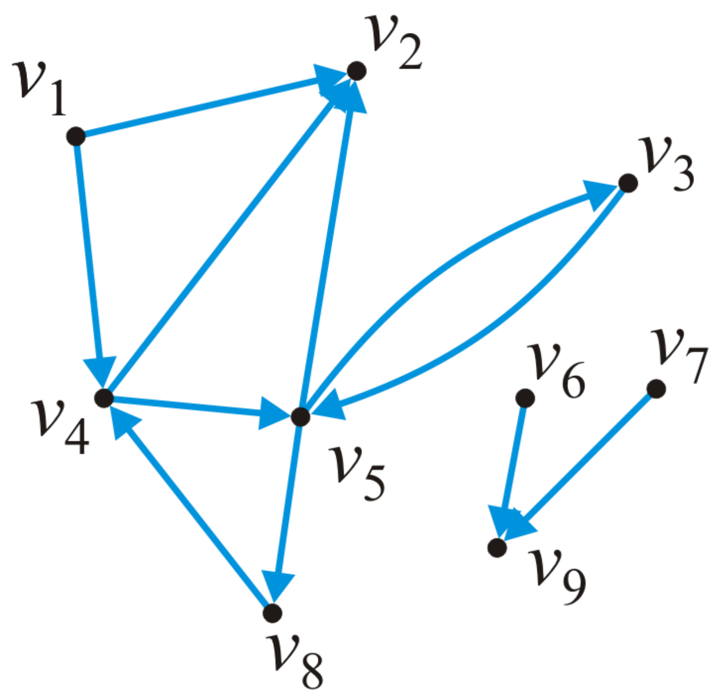


Adjacency matrix

Requiring more memory but also faster, an adjacency matrix

- The matrix entry (j, k) is set to true if there is an edge (v_j, v_k)

	1	2	3	4	5	6	7	8	9
1		T		T					
2									
3					T				
4		T			T				
5		T	T					T	
6									T
7									T
8				T					
9									



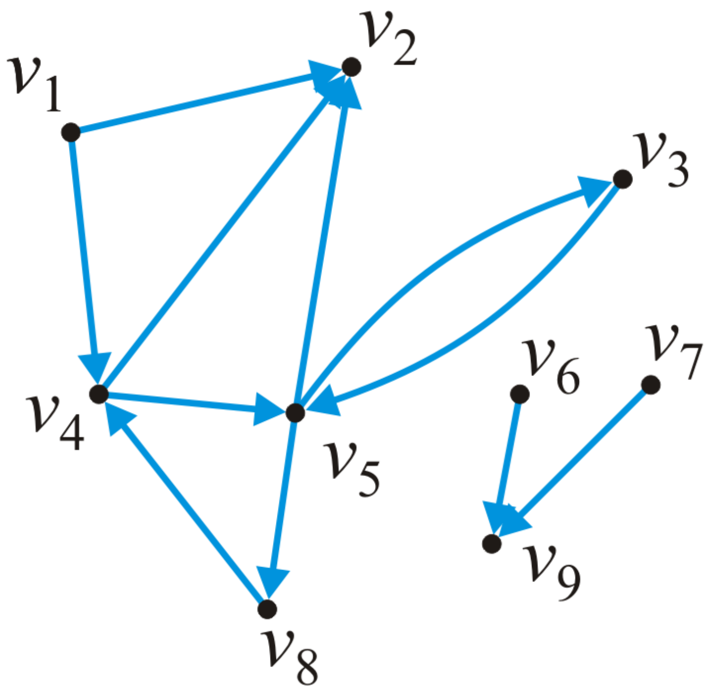
- Requires $\Theta(|V|^2)$ memory
- Determining if v_j is adjacent to v_k is $O(1)$
- Finding all neighbors of v_j is $\Theta(|V|)$

Adjacency list

Most efficient for algorithms is an adjacency list

- Each vertex is associated with a list of its neighbors

$$
\begin{array}{ll}
1 & \bullet \rightarrow 2 \rightarrow 4 \\
2 & \bullet \\
3 & \bullet \rightarrow 5 \\
4 & \bullet \rightarrow 2 \rightarrow 5 \\
5 & \bullet \rightarrow 2 \rightarrow 3 \rightarrow 8 \\
6 & \bullet \rightarrow 9 \\
7 & \bullet \rightarrow 9 \\
8 & \bullet \rightarrow 4 \\
9 & \bullet
\end{array}
$$



- Requires $\Theta(|V| + |E|)$ memory
- On average:
  - Determining if v_j is adjacent to v_k is $\mathrm{O}\left(\frac{|E|}{|V|}\right)$
  - Finding all neighbors of v_j is $\Theta\left(\frac{|E|}{|V|}\right)$

Adjacency Matrix vs Adjacency List

Two common graph operations:

1. Determine whether there is an edge from vertex i to vertex j.
2. Find all vertices adjacent to a given vertex i.

An adjacency matrix supports operation 1 more efficiently.

An adjacency list supports operation 2 more efficiently.

An adjacency list often requires less space than an adjacency matrix.

- Adjacency Matrix: Space requirement is $O(|V|^2)$
- Adjacency List : Space requirement is $O(|E| + |V|)$, which is linear in the size of the graph.
- Adjacency matrix is better if the graph is dense (too many edges)
- Adjacency list is better if the graph is sparse (few edges)

Some Definitions

Simple Graph: no self loop + no parallel edge

Null Graph: a graph without any edge

Isolated vertex: degree(v_i) = 0

Pendent vertex: degree(v_i) = 1

Complete graph: each vertex connected with every vertices. Represented by K_n

Bipartite graph: Represented by $K_{n,m}$ .Complete bipartite graph

Regular graph: same degree of each vertex.

Cycle graph: regular graph + degree 2 of each vertex. Represented by C_n

Wheel Graph: The graph obtained from C_{n-1} by joining each vertex to a new vertex v is the wheel on n vertices denoted by W_n

Euler Graph & Hamiltonian Graph

If some closed walk in a graph contains all the edges of the graph, then the graph is an euler graph.

A given connected graph G is an euler graph if and only if all vertices of G are of even degree.

Semi eulerian: if it has exactly two vertices of odd degree.

A closed walk that traverses every vertex of graph G exactly once, except the starting vertex is known as Hamiltonian circuit.