Foundations of Semantic Knowledge Graphs

Prof. Dr. Stefan Linus Zander

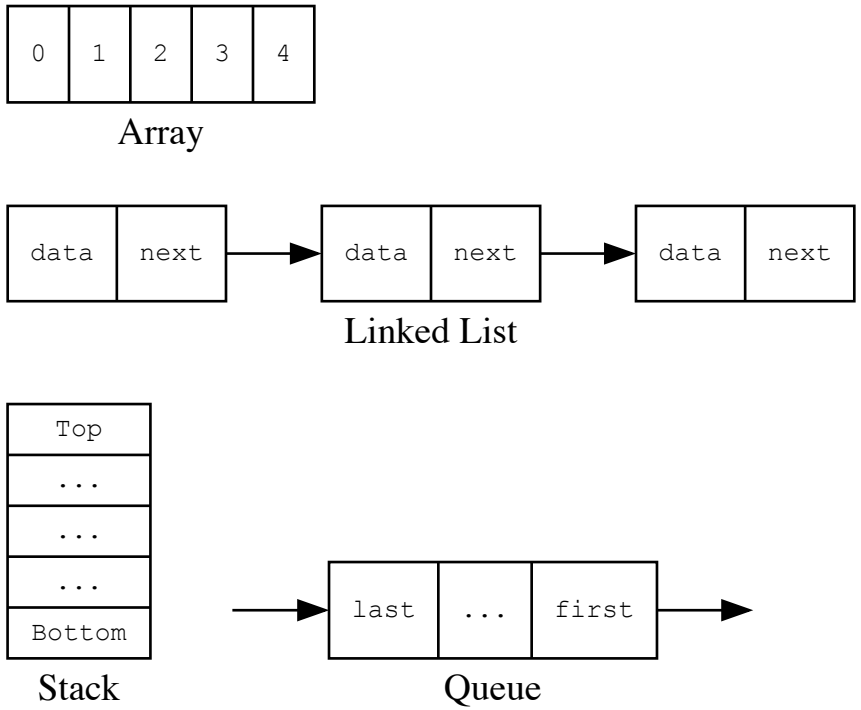
A Mathematical Introduction to Graphs in Computer Science

What you will learn in this Unit

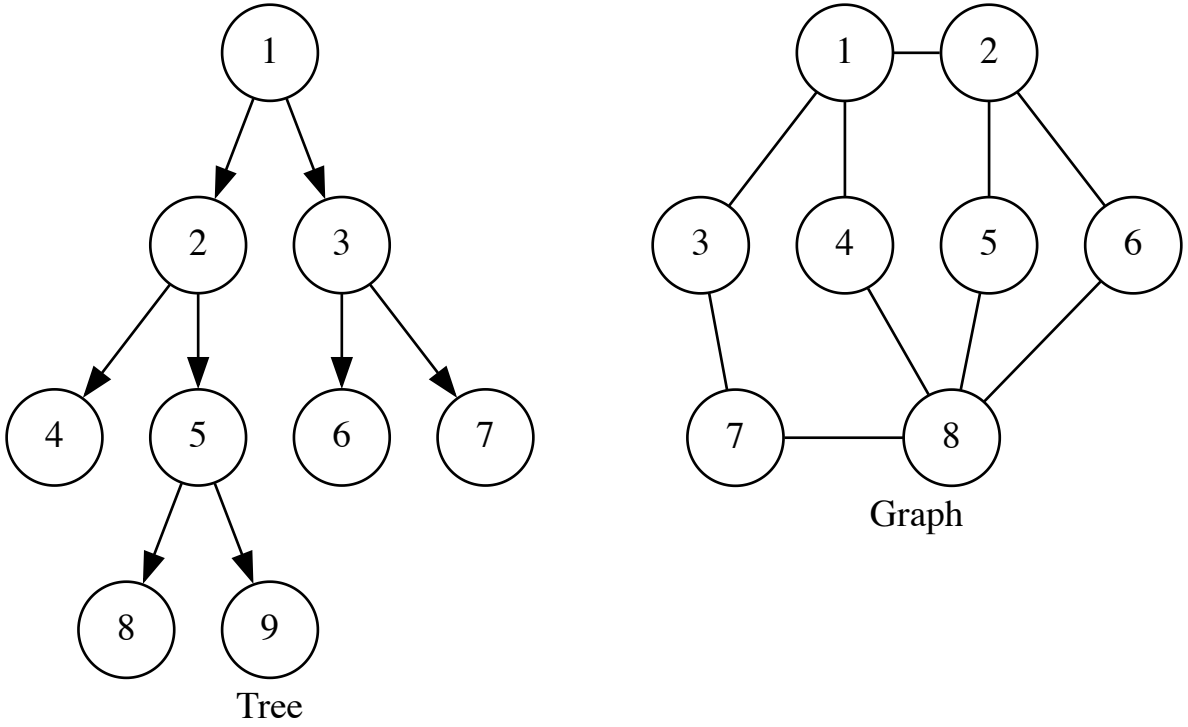
We will discuss the mathematical foundation underlying knowledge graphs.

There are a number of different data structures for storing data

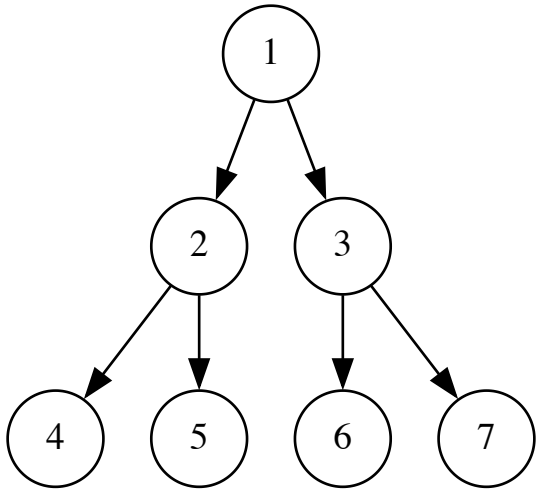
Linear Data Structures



Non-linear Data Structures

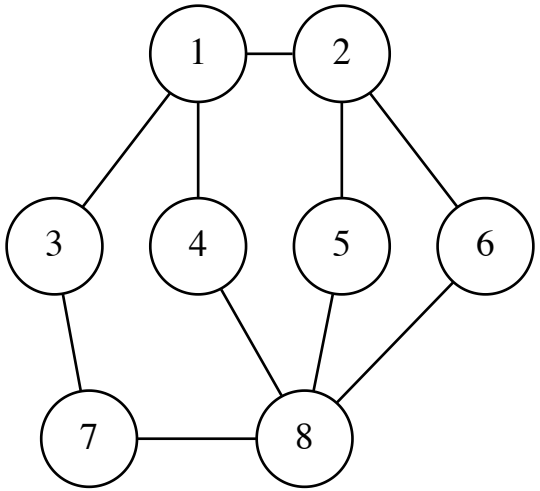


Trees and Graphs share a number of common attributes



Tree

- A tree with N nodes has $N - 1$ edges
  - one edge for each parent-child relationship
- All nodes in a tree except the root have exactly one parent
- All nodes must be reachable from the root through exactly one defined path
- Any tree is also a graph



Graph

- A graph consists of a set of nodes and edges
- In Graph Theory, nodes are usually called vertices
- No rules that dictate the connection among nodes, ie., edges can be connected in any possible way
- CS usually adapts the mathematical notion of graphs
- CS studies graphs and related algorithms in Graph Theory

A Mathematical Definition of Graphs

Graph

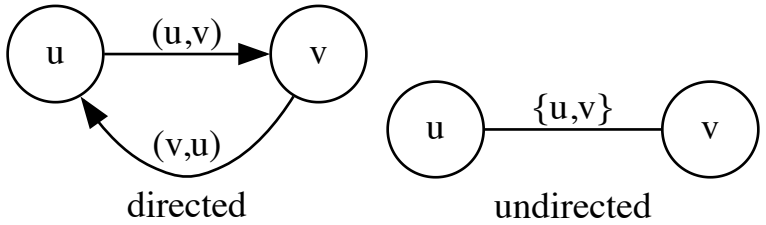
A graph G is an ordered pair of a set V of vertices and a set E of edges.

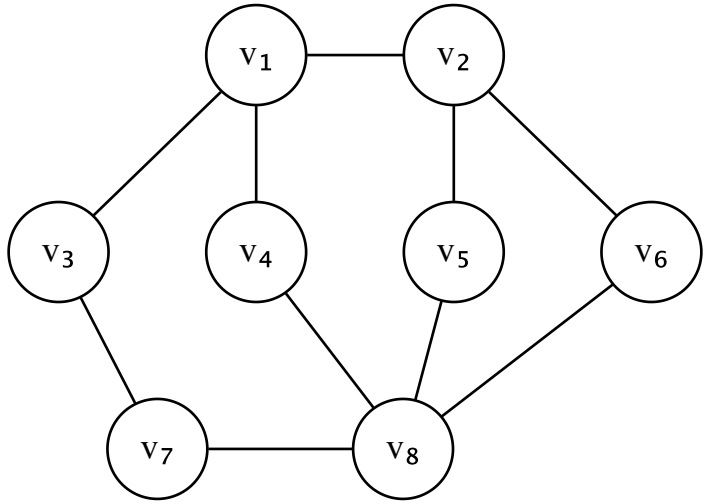
$$G = (V, E)$$

Ordered pair: $(a, b) \neq (b, a) \quad \text{if } a \neq b$

Unordered pair: $\{a, b\} = \{b, a\}$

Edges

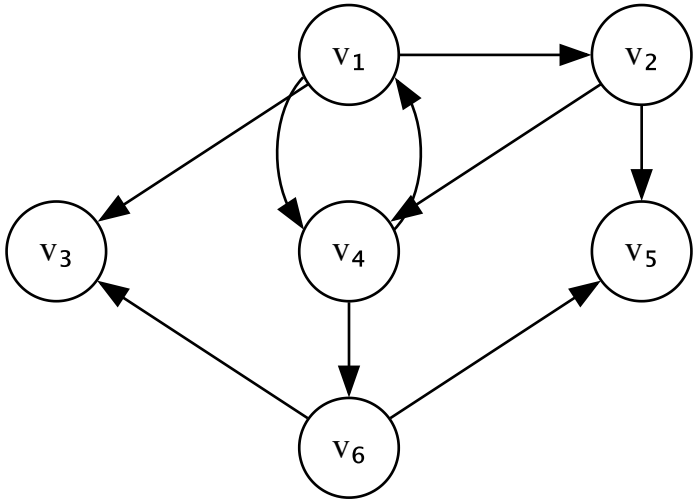




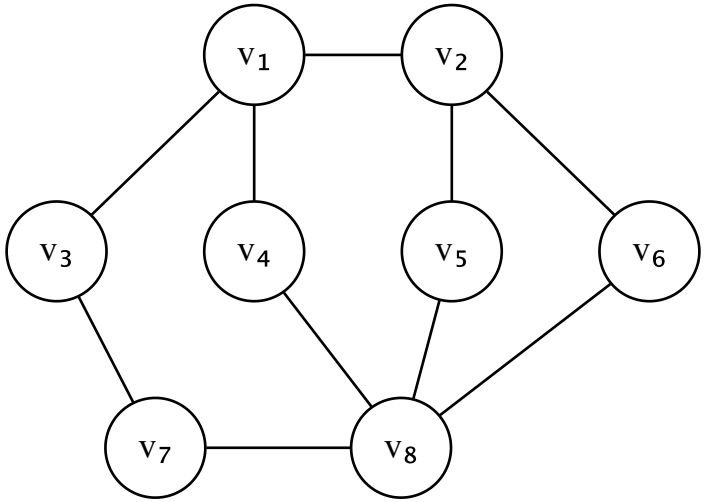
$$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$$

$$E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_5\}, \{v_2, v_6\}, \{v_3, v_7\}, \{v_4, v_8\}, \{v_5, v_8\}, \{v_6, v_8\}, \{v_7, v_8\}\}$$

Directed vs. Undirected Graphs



a directed graph or Digraph

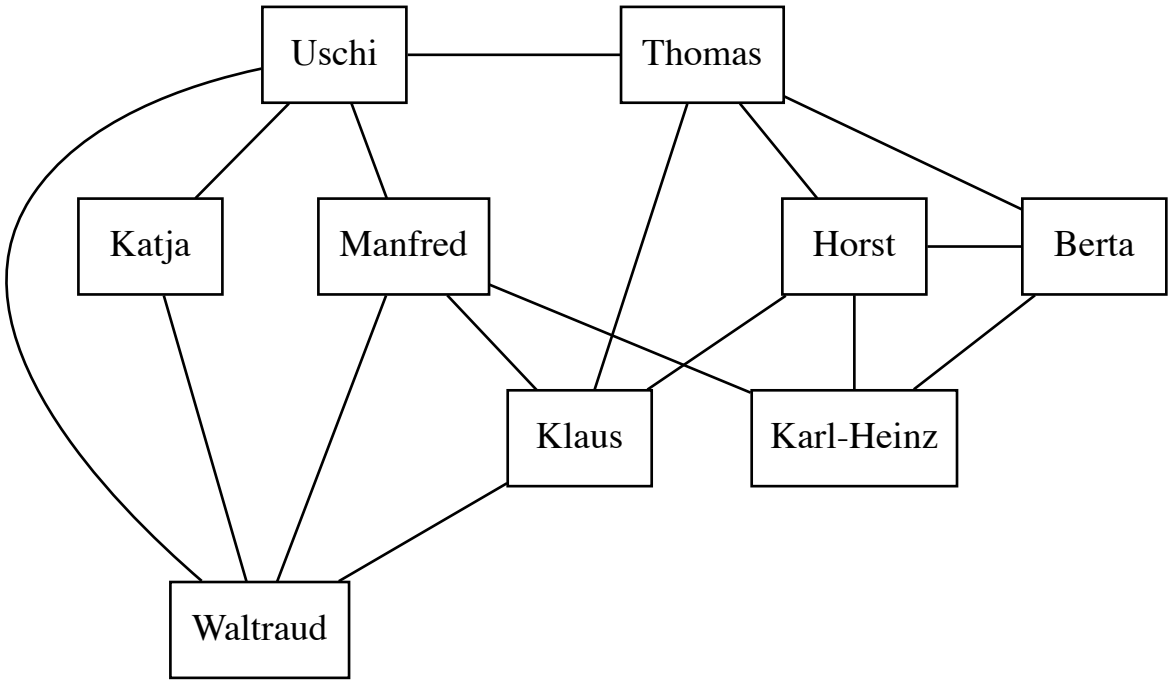


an undirected graph

Remarks

- Many real-world systems or problems can be modelled using a graph.
- Graphs can be used to represent any collection of objects having some kind of pairwise relationship.

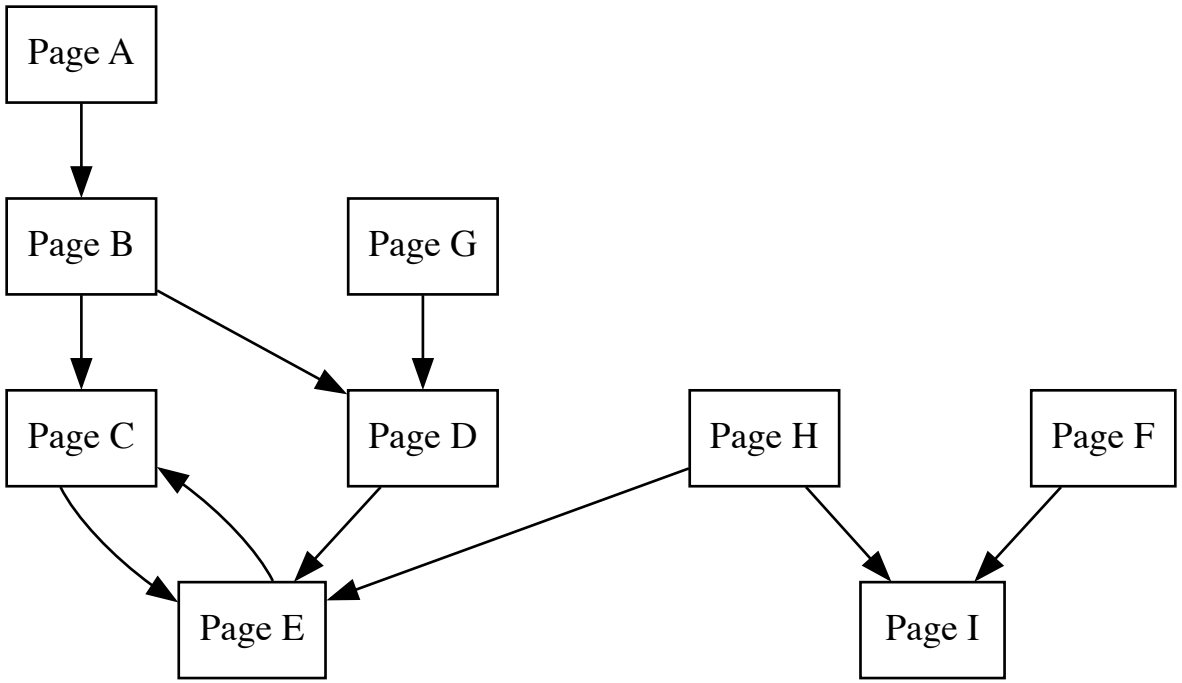
Example Graphs: Social Network



A social network graph

- **Social networks** are usually represented as undirected graphs since friendships are mutual relationship
- Vertices represents persons and edges represent friend relationships
- A lot of problems can be solved by applying standard algorithms from graph theory
  - e.g. suggesting friends of friends who are not connected
  - e.g. can we suggest some friends for 'Katja'?
- Standard graph problem:
  Find all nodes having a shortest path length of '2' from 'Katja'

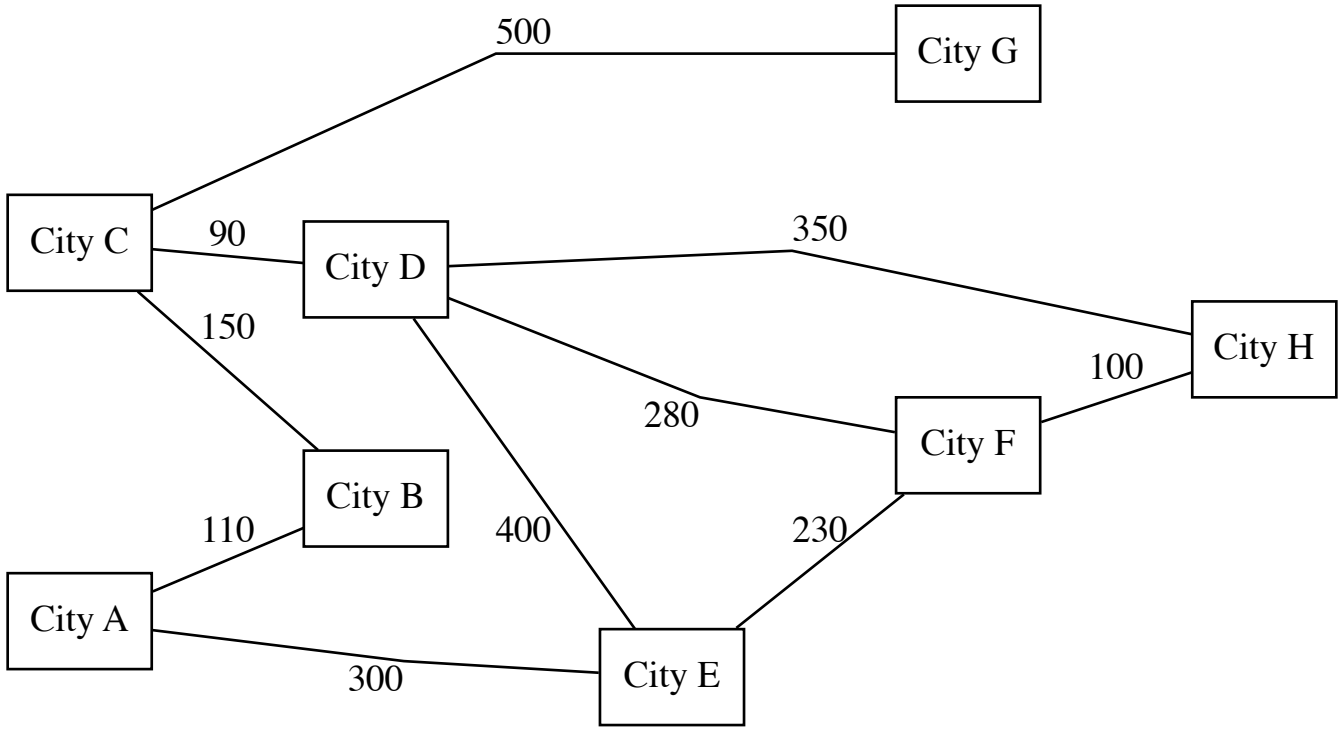
Example Graphs: Interlinked Web Pages



link relations between web pages

- A vertice represents a web page with a unique URL
- An edge represents a link from one page to another
- The edges are directed because the relationship is not mutual
- Application Scenario: Web Crawling to collect and store data about web pages
- Web crawling is basically graph traversal
- Standard graph probelm: **Centrality** (degree, betweenness, closeness, eigenvector, etc)

Example Graphs: Road Networks



Intercity road network (assuming that all highways are bidirectional)

Problem: What is the shortest direction from 'City F' to 'City B' ?

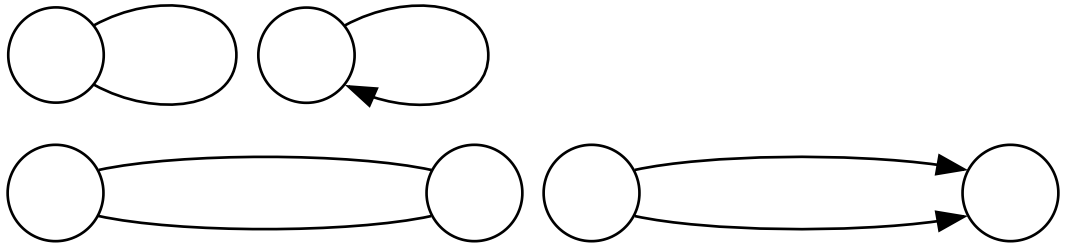
- With weights added to connections we can accumulate the single weigths and find the shortest route

- Edges represent bidirectional intercity connections
- Associating a weight or cost to a connection accounts for their length
- Values associated with edges are denoted as weight or cost
- Such graphs are called weighted graphs
- Connections must be treated differently due to their varying lengths
- Road networks can be represented as weighted undirected graphs
- Some application scenarios require a different treatment of edges
- All graphs can be treated as weighted graphs

Special edge types

Self-Loop

Multi-Edge

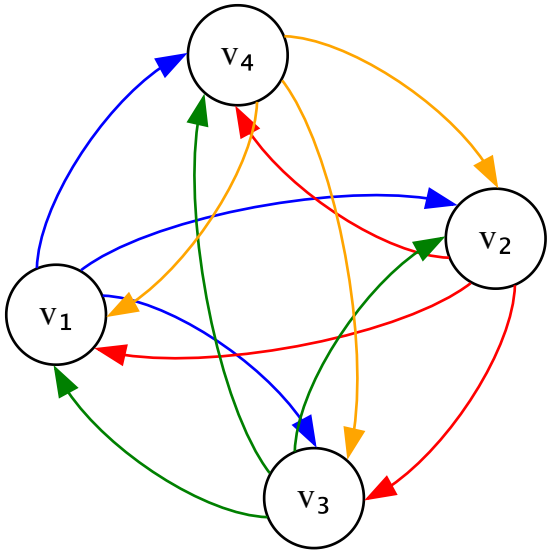


A graph is called a multi-edge graph if it contains pair of vertices that are connected by more than one edge.

If there are no self-loops or multi-edges in a graph, it is a simple graph.

Properties of Graphs (2/2)

Number of Edges



$$V = \{v_1, v_2, v_3, v_4\}$$
$$|V| = 4$$

if $|V| = n$ then

$$0 \leqq |E| \leqq n(n-1), \qquad \text{if directed}$$
$$0 \leqq |E| \leqq \frac{n(n-1)}{2}, \qquad \text{if undirected}$$

Assuming no self-loop or multi-edge.

$$\text{if } |V| = 10, \qquad |E| \leqq 90$$
$$\text{if } |V| = 100, \qquad |E| \leqq 9900$$

A graph is dense, if the number of edges is close to its max.
A graph is sparse, if the number of edges is close to $|V|$.

However, there is no defined boundaries for dense and sparse, it depends on the context.

This classification is important, since a lot of decisions are made based on whether the graph is dense or sparse (e.g. choosing a different storage structure in computer's memory for dense graphs (ie adjacency matrix vs. adjacency list)).

Properties of Graphs: Paths, Walks, and Trails

In graph theory there is some inconsistency between the terms path and walk.

A walk is a sequence of vertices where each adjacent pair is connected by an edge.

$$< v_1, v_2, v_6, v_8, v_5, v_2, v_1, v_4 >$$

A path is a walk with no repeated vertex

$$< v_1, v_2, v_6, v_8, v_7, v_3 >$$

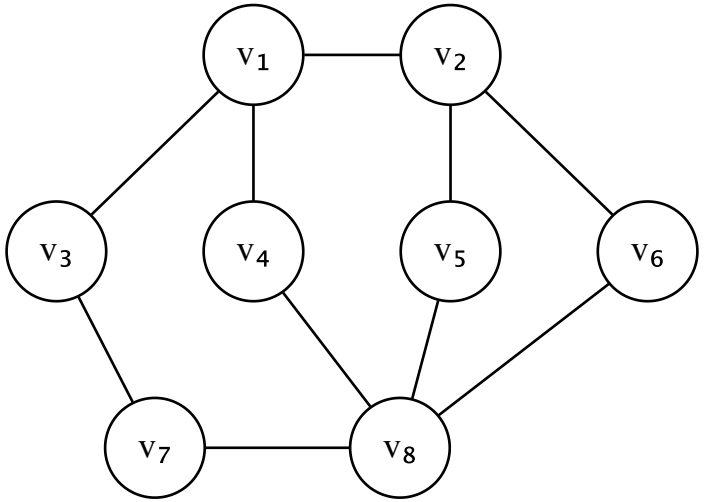
A simple path is a walk in which no vertices and edges are repeated.

$$< v_1, v_2, v_6, v_8 >$$

A trail is a walk with no repeated edge (vertices can be repeated)

$$< v_1, v_2, v_5, v_8, v_4, v_1, v_3 >$$

The length of a walk trail, path or cycle is its **number of edges**.



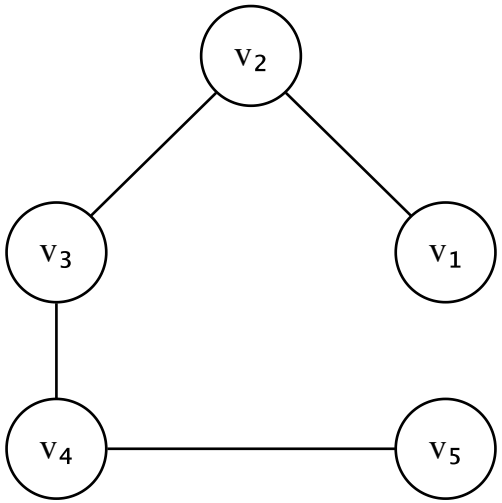
an undirected graph

If any other path is possible, there must be a simple path $\Leftrightarrow$ if there is a simple path, any other "path" is also possible

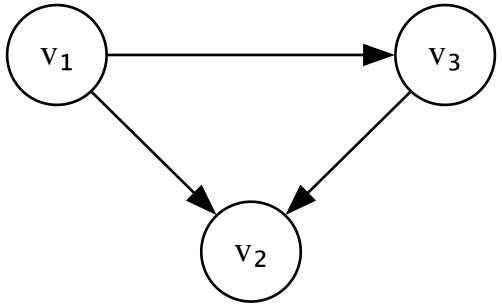
Walk and path are often used as synonyms but most often when we say path we mean simple path, a path in which vertices and edges are not repeated.

Properties of Graphs: Strongly Connected Graphs

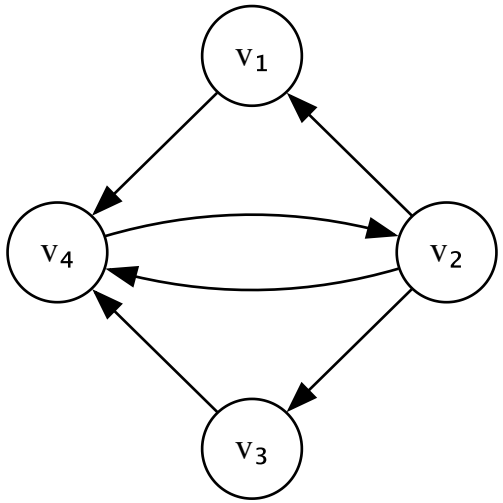
A **graph** is a strongly connected graph if there is a path from any vertex to any other vertex.



A connected graph



A weakly connected graph



A strongly connected graph

The degree and type of **connectedness** is an important concept in graph theory.

Properties of Graphs: Cycles

A **closed walk** starts and ends at the same vertex and its length is > 0.

$$< v_1, v_2, v_5, v_8, v_4, v_1 >$$

A (simple) **cycle** is a closed walk with no repetition other than start and end.

$$< v_1, v_2, v_5, v_8, v_4, v_1 >$$

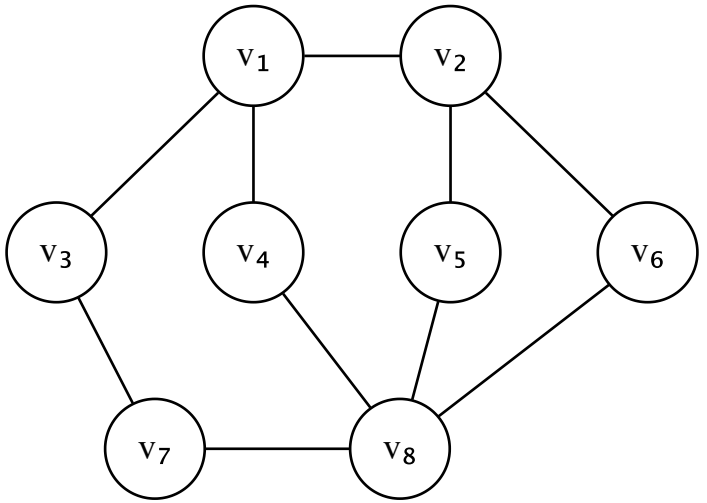
An **acyclic graph** is a graph with no cycles.

A tree with undirected edges would be an undirected acyclic graph

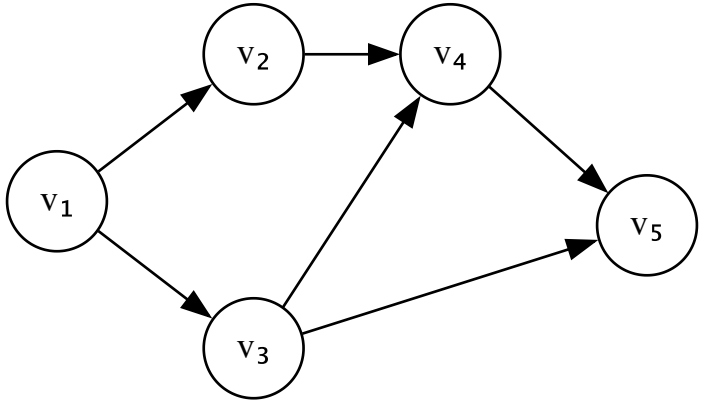
A tree would not have a simple cycle.

A **directed acyclic graph** is often called DAG.

Common problem in DAGs: Finding the shortest route from one vertice to another.



an undirected graph



A directed acyclic graph (DAG)

Graph Representation

Representing graphs: Adjacency Matrix

There are several ways for representing graphs in Computer Science.

Definition The adjacency matrix of a graph $G = \langle V, E \rangle$ is the boolean $|V| \times |V|$ matrix that contains, at any coordinate $\langle v_1, v_2 \rangle$, the value 1 if there is an edge connecting v_1 and v_2.

Source: Introduction to Knowlege Graphs, Prof. Dr. Markus Krötzsch, TU Dresden

Notes

- Adjacency matrices for undirected graphs are symmetric.
- Loops (if allowed) show up as 1 in the diagonal.
- The matrix could be adapted to multi-graphs by storing the numbers of edges.
- The matrix could be adapted to labelled simple graphs by storing the labels.

Representing graphs: Adjacency List

There are several ways for representing graphs in Computer Science.

Definition The adjacency list of a graph $G = \langle V, E \rangle$ is the list of all of its edges.

Source: Introduction to Knowlege Graphs, Prof. Dr. Markus Krötzsch, TU Dresden

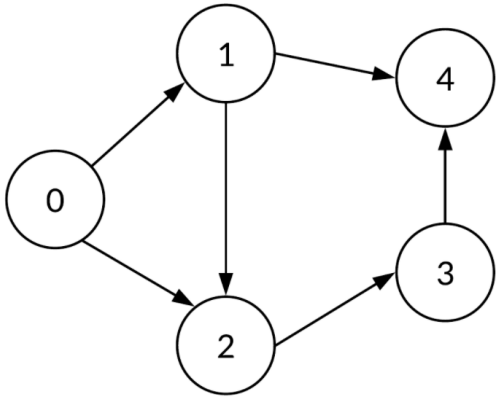
Notes

- We can write edges as pairs (order is irrelevant for undirected graphs)
- Loops (if allowed) show up as edges with repeated vertices
- The list could be adapted to multi-graphs by adding the number of edges to each line, or by allowing repeated lines
- The list could be adapted to labelled graphs by adding labels to each line (for multi-graph: repeat lines rather than also storing number).
- The list does not encode V: vertices without edges are missing (might be listed separately if relevant to application)

Each graph representation has its pros 👍 and cons 👎

Adjacency Matrix

- 👍 space efficient for dense graphs (1 bit per edge)
- 👍 can be processed with matrix operations (highly parallel)
- 👎 space inefficient for sparse graphs
- 👎 not natural for labelled multi-graphs

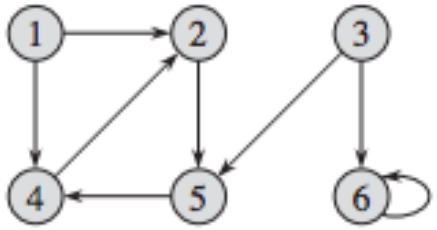


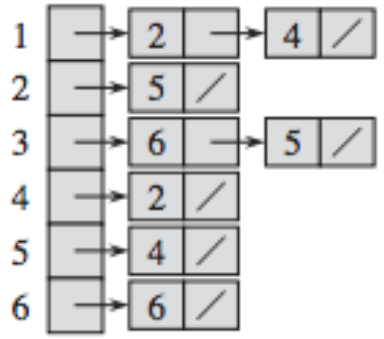
Adjacency Matrix

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

Adjecency List

- 👍 space efficient for sparse graphs;
- 👍 easy to use for labelled multi-graphs;
- 👎 harder to process (esp. if edge order can be random)
- 👎 not space efficient for dense graphs





Note: **Knowledge graphs** are typically **sparse** and **labelled**, but parallel processing still makes matrices attractive in some applications.

Source: Lecture Slides of Prof. Dr. Markus Krötzsch, TU Dresden; Pictures taken from https://www.cs.mtsu.edu/~xyang/3080/

Graph representations in mathematics are not sufficient for knowledge graph representations in Computer Science

We have seen that graphs can be encoded in several ways:

- Adjacency matrix (and variants)
- Adjacency list (and variants)
- Other derived representations

This works for storing and manipulating graphs in software, but it is insufficient to encode meaning and exchange graphs across apps.

Open questions:

- What kind(s) of graph do we want to exchange ?
- How are vertices given (numbers? strings? specific ids? ...) ?
- Are edge labels supported and what are they ?
- Can the graph include values of data types (integer? float? string? times? ...)?
- How exactly are these things encoded in bytes in a file?

Source: Compiled from Lecture Materials about Knowledge Graphs from Prof. Dr. Markus Krötzsch, TU Dresden