CS2521: Graphs

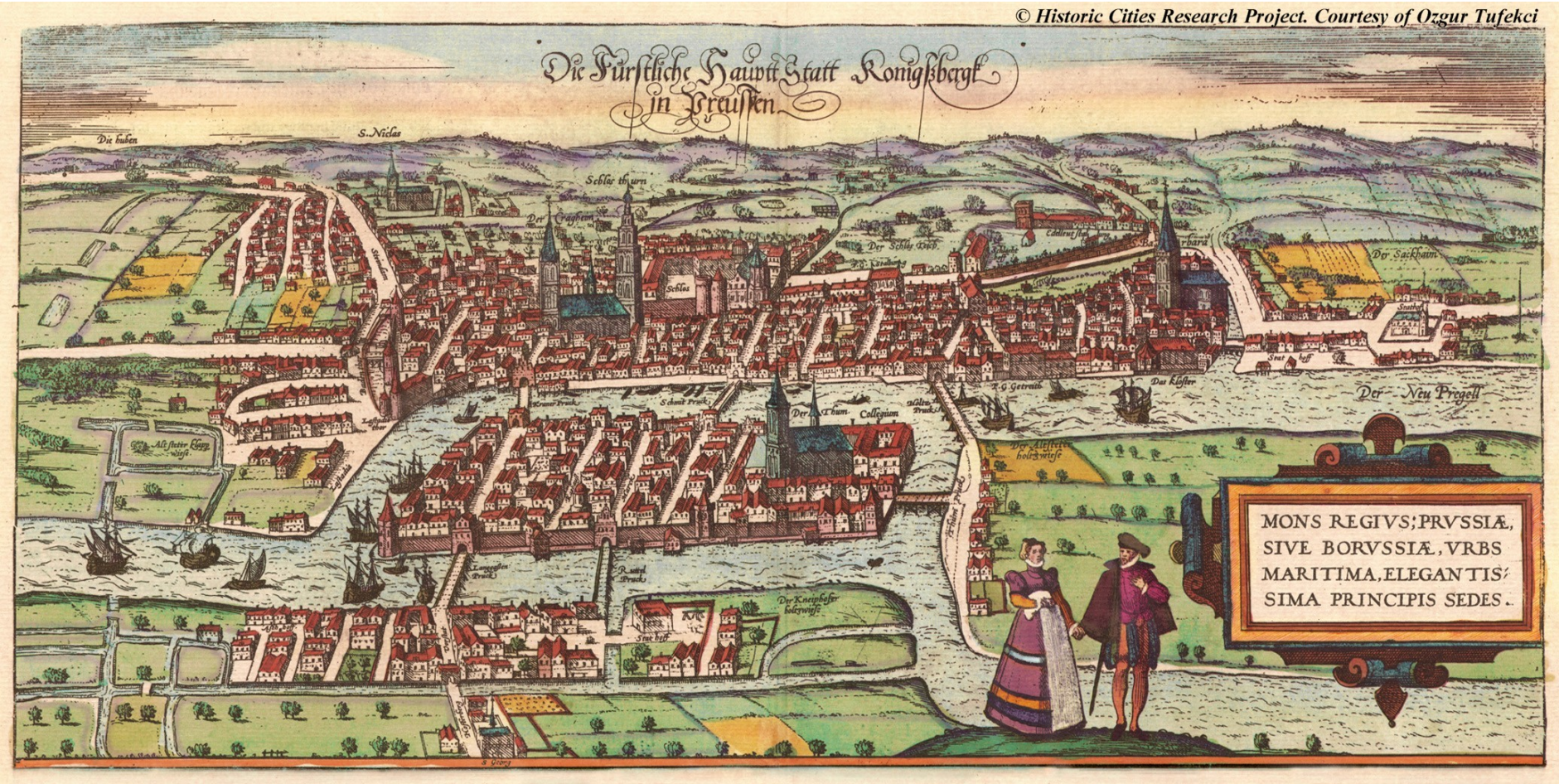
N. Oren
n.oren@abdn.ac.uk

University of Aberdeen

Overview

- What is a graph?
- How do we encode graphs, and why choose one encoding over another?
- Graph related concepts
- How do we use graphs?

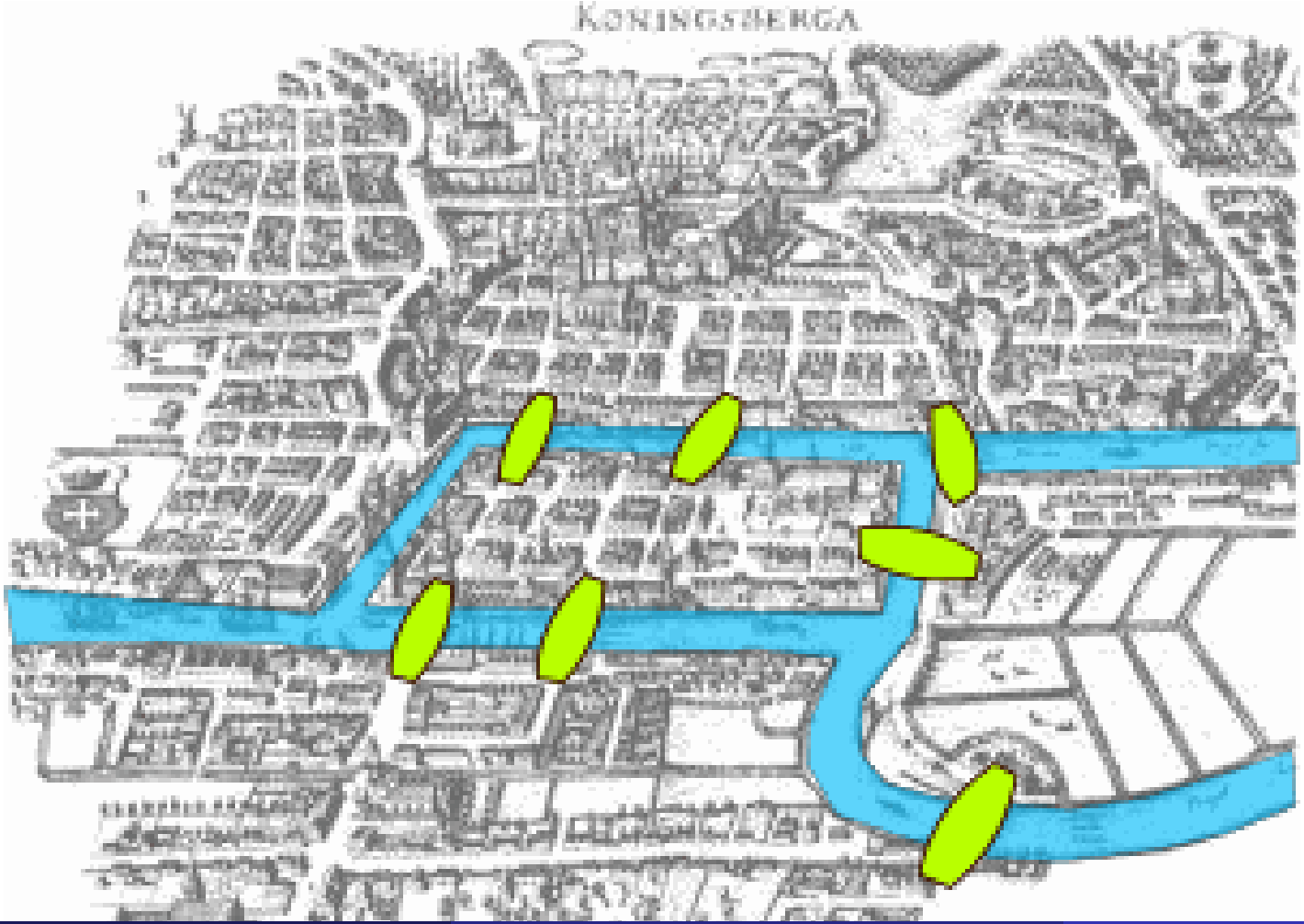
A bit of history - The bridges of Konigsberg



Can you find a way to walk through the city, going over each bridge only once?

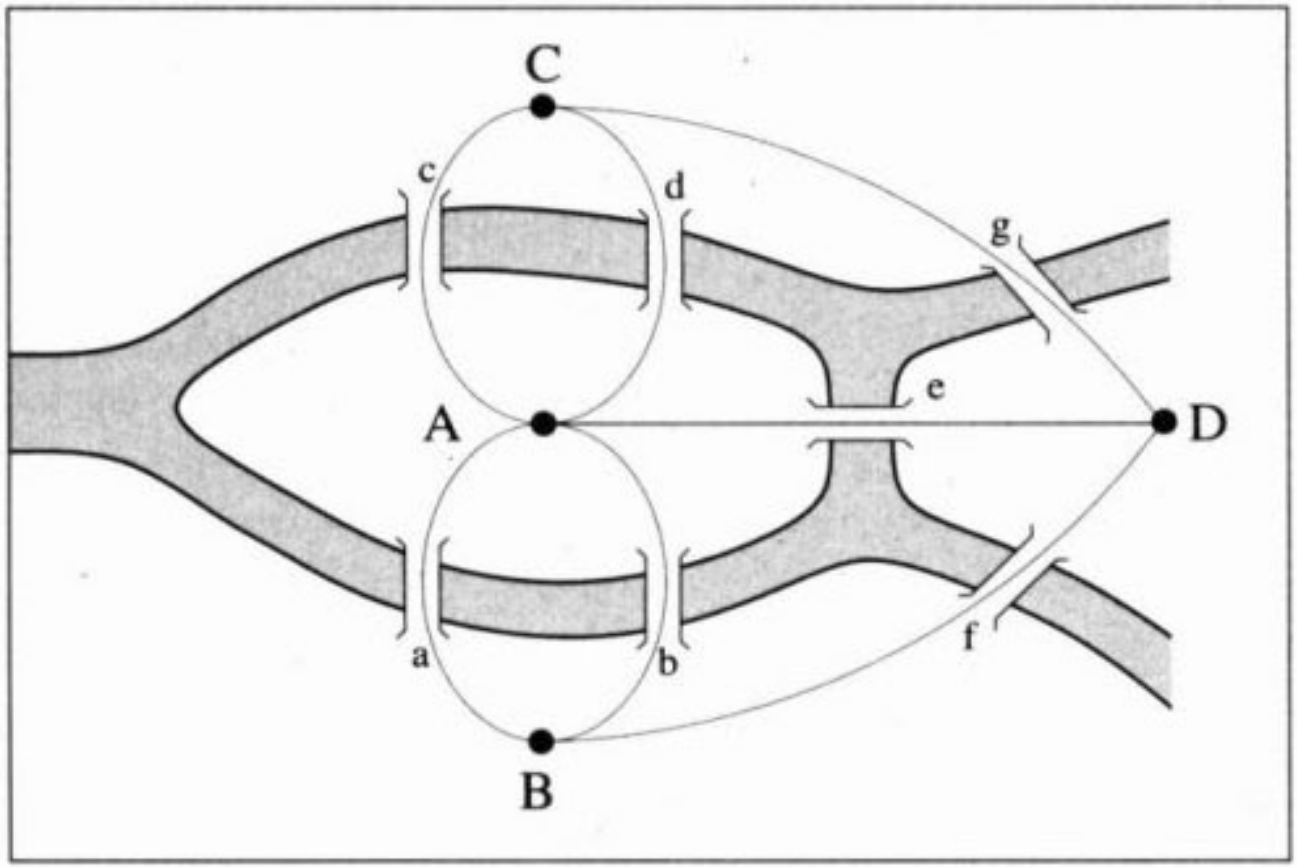
A bit of history - The bridges of Konigsberg

- Simplify the problem:
  - All we care about is being on the same landmass as the start of the next bridge we're crossing.
  - So we can remove roads.



A bit of history - The bridges of Konigsberg

- The abstract structure on which the analysis is based is called a graph.



- A graph consists of a set of nodes or vertices, and edges.
- Rather than exhaustively searching for a solution, we can formalise the problem and answer it by reasoning about the graph's properties.

A bit of history - The bridges of Konigsberg

- This analysis was performed by Leonhard Euler in 1736.
  - Consider a point not at the start or end of the tour. We must have reached the landmass, and also leave it.
  - So in the middle, we need an even number of bridges (so as not to get stuck).
  - At the start, we can leave but not return, and similarly at the finish.
  - So we can also have zero or two landmasses with an odd number of bridges.
  - Why can't we end up with one landmass with an odd number of bridges?
  - But problem has odd number of bridges, so no solution exists.
- Generally, the problem is to find a sequence of adjacent edges, starting and finishing at the same vertex, using each edge exactly once. This is called an Eulerian circuit.

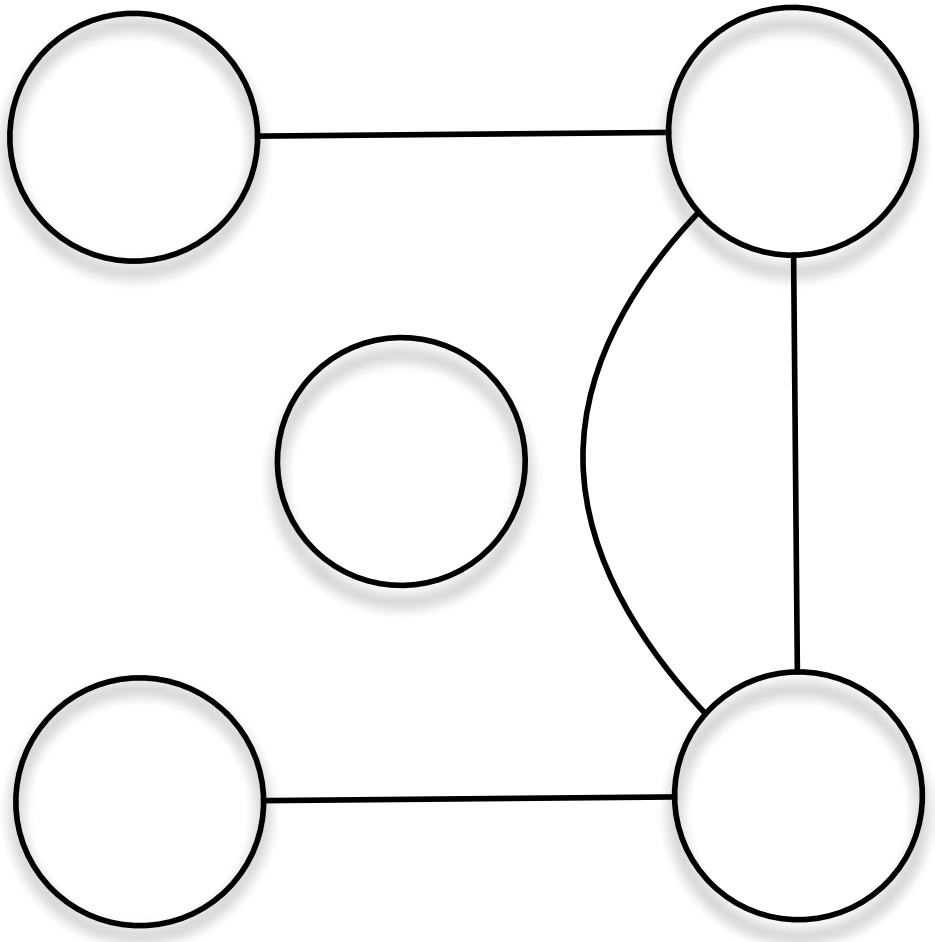
Formalisation

- A directed graph is a pair (V, E) where V is a set of vertices, and E is a set of edges such that for any edge $(s, t) \in E$, $s, t \in V$.
- Note
  - The vertices within the edge represent the edge's ends.
  - For any edge, it could be the case that $s = t$, in which case a loop exists.

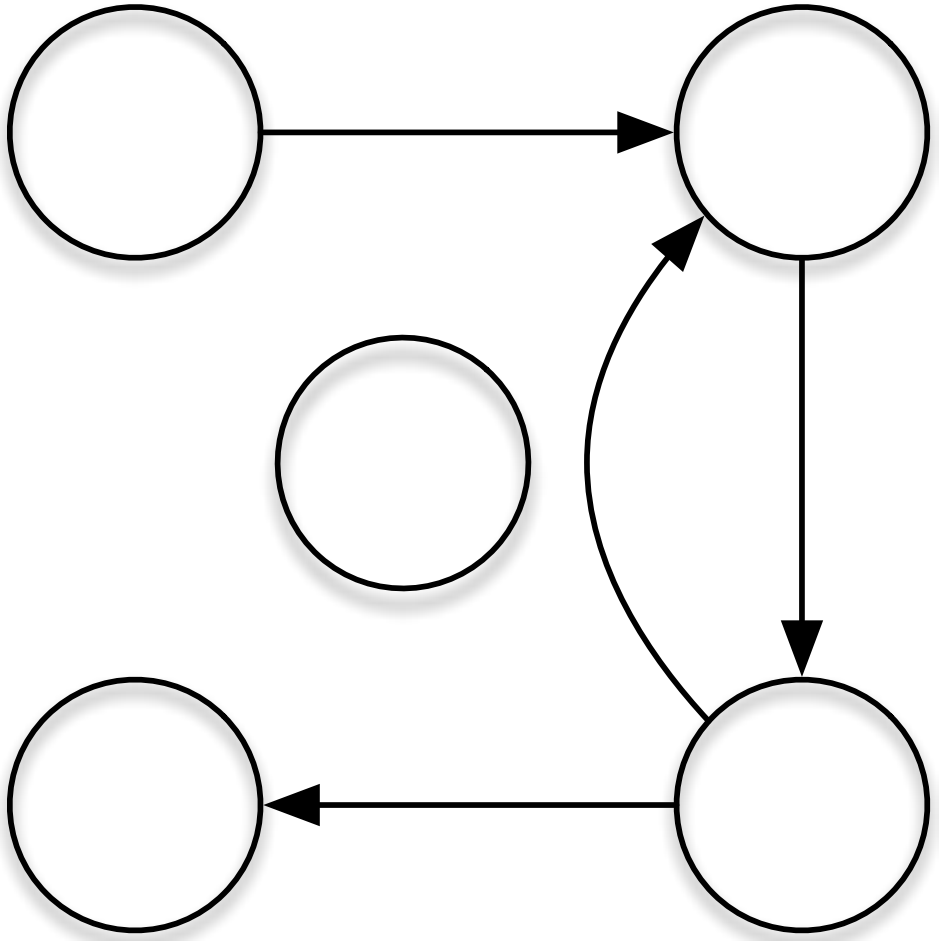
Formalisation

- An undirected graph is a pair (V, E) where V is a set of vertices, and E is a set of edges such that for any edge $\{s, t\} \in E$, $s, t \in V$.
- Note
  - Now the edge is a set of two vertices, meaning we cannot distinguish a start vertex from an end vertex.

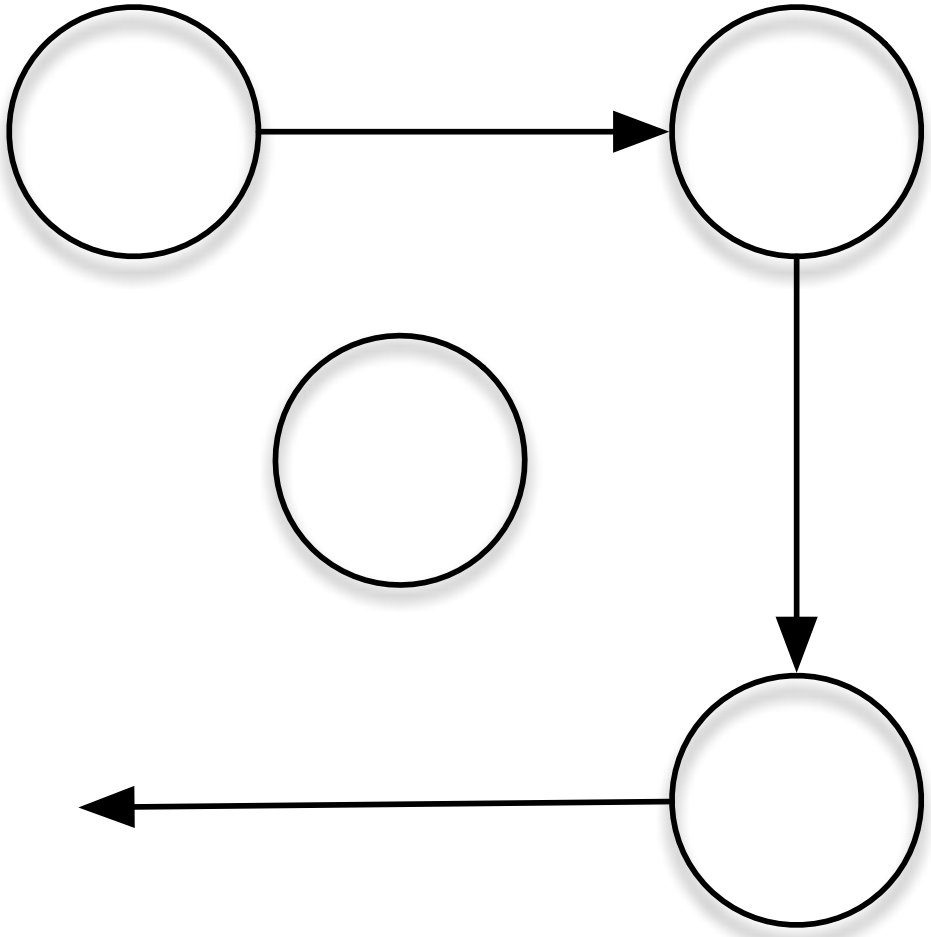
Examples



Examples

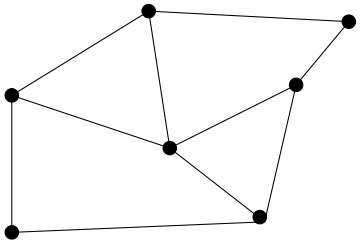


Examples

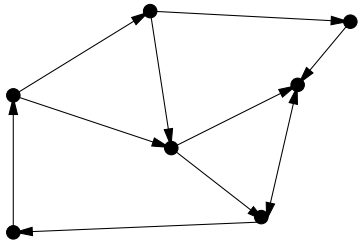


Examples

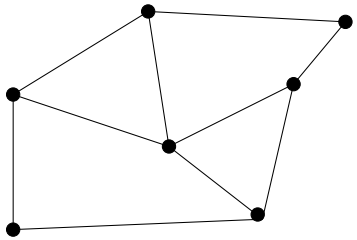
Other types of graphs



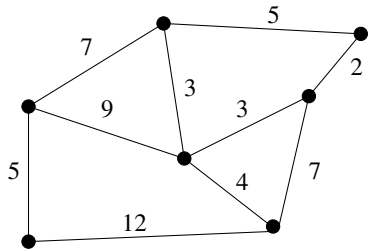
undirected



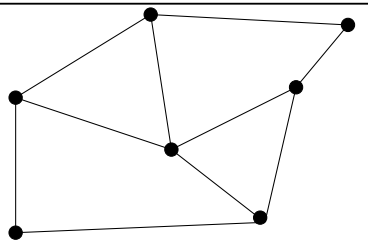
directed



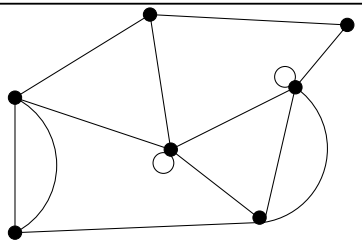
unweighted



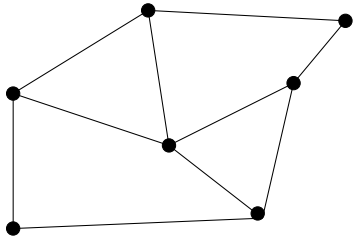
weighted



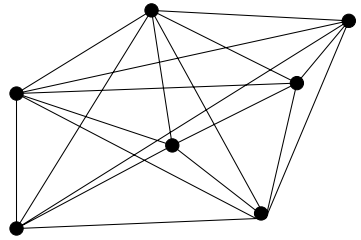
simple



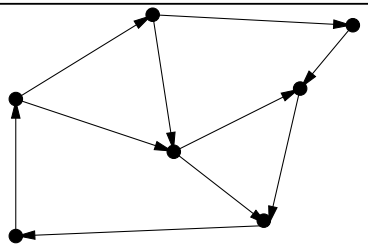
non–simple



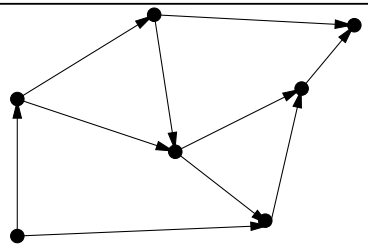
sparse



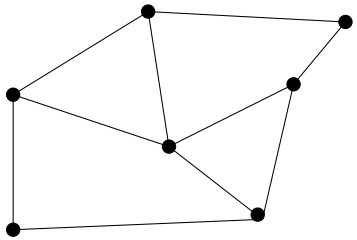
dense



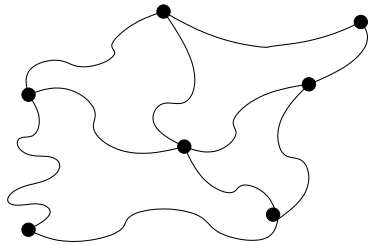
cyclic



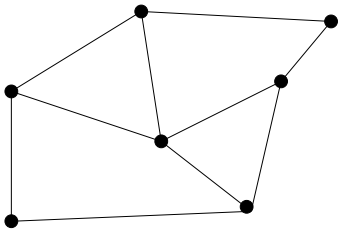
acyclic



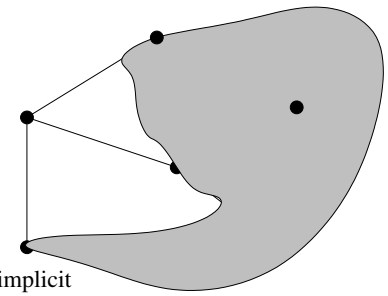
embedded



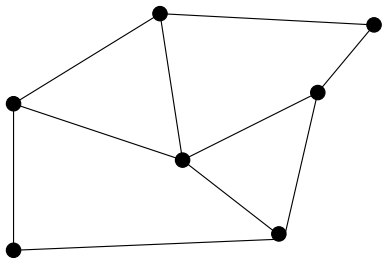
topological



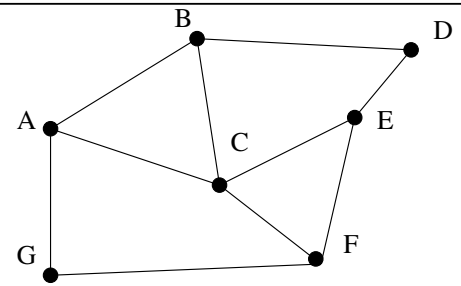
explicit



implicit



unlabeled



labeled

Other types of graphs

- Simple: no multi-edges or loops (makes things simpler to implement)
- Sparse: only a small fraction of vertex pairs (edges) exist.
- Acyclic: no cycles
- Embedded: geometric position of vertices and edges have meaning. Otherwise, topological
- Implicit/explicit: Is the graph fully represented, or is it conceptual?
- Labelled/Unlabelled: Each vertex is associated with a unique identifier.

Social Graphs

- Imagine you were building a social network application.
  - How are graphs useful in such a setting?
- What are the properties of these graphs?

Social Graphs

- Imagine you were building a social network application.
  - How are graphs useful in such a setting?
- What are the properties of these graphs?
  - Is friendship two way? (directed/undirected)
  - Are you your own friend? (simple)
  - Can you have different types of friends (multi-edge)
  - How good a friend are you? (weighted)
  - Who has the most friends? (degree of a vertex)

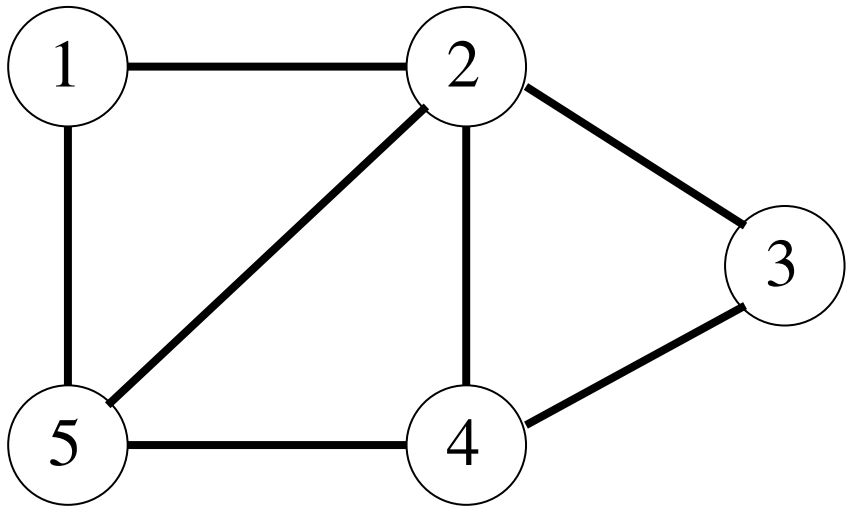
Graph Representation

- We can store a graph in different ways.
- Most commonly, we use either an adjacency matrix or adjacency list.

Adjacency Matrices

- Given a graph (V, E), create a $|V| \times |V|$ matrix of 0s, and place a 1 at position i, j if $(i, j) \in E$

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



Adjacency Matrices

- How easy is it to check if (i, j) is an edge?

Adjacency Matrices

- How easy is it to check if (i, j) is an edge? $O(1)$
- How easy is it to update or delete an edge?

Adjacency Matrices

- How easy is it to check if (i, j) is an edge? $O(1)$
- How easy is it to update or delete an edge? $O(1)$
- So what's the downside?

Adjacency Matrices

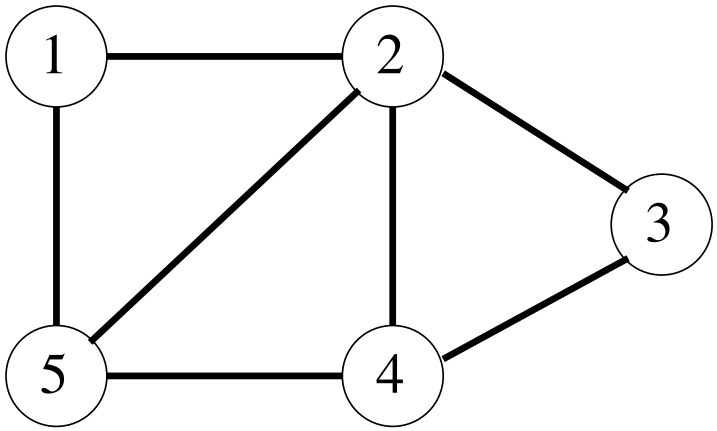
- How easy is it to check if (i, j) is an edge? $O(1)$
- How easy is it to update or delete an edge? $O(1)$
- So what's the downside?
- Unless your graph is dense, most entries of the matrix will be 0. Nevertheless, you need $O(|V|^2)$ memory to store the graph.
- Facebook: $\sim 10^9$ users. Memory requirements: 10^{18} (a petabyte is 10^{15}).

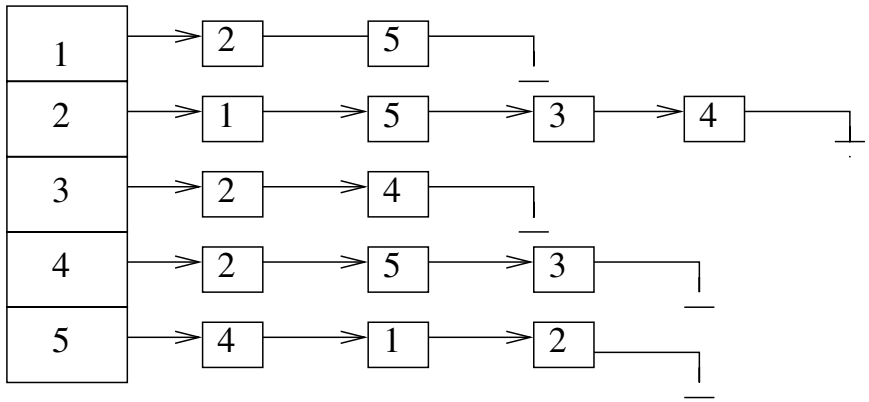
Dense vs Sparse

- A complete graph has every vertex connected to every other.
- If most vertices are not connected, the graph is sparse.
- A graph is dense if it is not sparse.
- An array is sparse if most entries have identical (default) values (normally 0).
- Many CS problems involve dealing with sparse data.

Adjacency Lists

- For each vertex, store a list of vertices to which it connects.





Implementation

- Most common implementations use a (singly) linked list.
- Linked lists
  - A set of nodes containing a value and a pointer to the next node.
  - A special null pointer signifies the end of the list.

Adjacency Lists

- How do we check if (i, j) is an edge?

Adjacency Lists

- How do we check if (i, j) is an edge?

  L = linked list for vertex i
  repeat
    l=Get next element of L
    if $l == j$ **then**
      return true
    end if
  until $l =$ `null`
  return false

- If v is the vertex with highest degree (say d) for which w is not present, then complexity is $O(d)$.

Adjacency List vs Adjacency Matrix

- Print all edges of a graph

Require: L elements in the adjacency list
function PrintGraph(L)
  for all $l \in L$ **do**
    while $l \neq$ `null` **do**
      m=succ(l)
      print(l,m)
      l=m
    end while
  end for
end function

Require: M adjacency matrix
function PrintGraph(M)
  for all $cin|M|$ **do**
    for all $r \in |M|$ **do**
      if $M[c, r] == 1$ **then**
        print(c,r)
      end if
    end for
  end for
end function

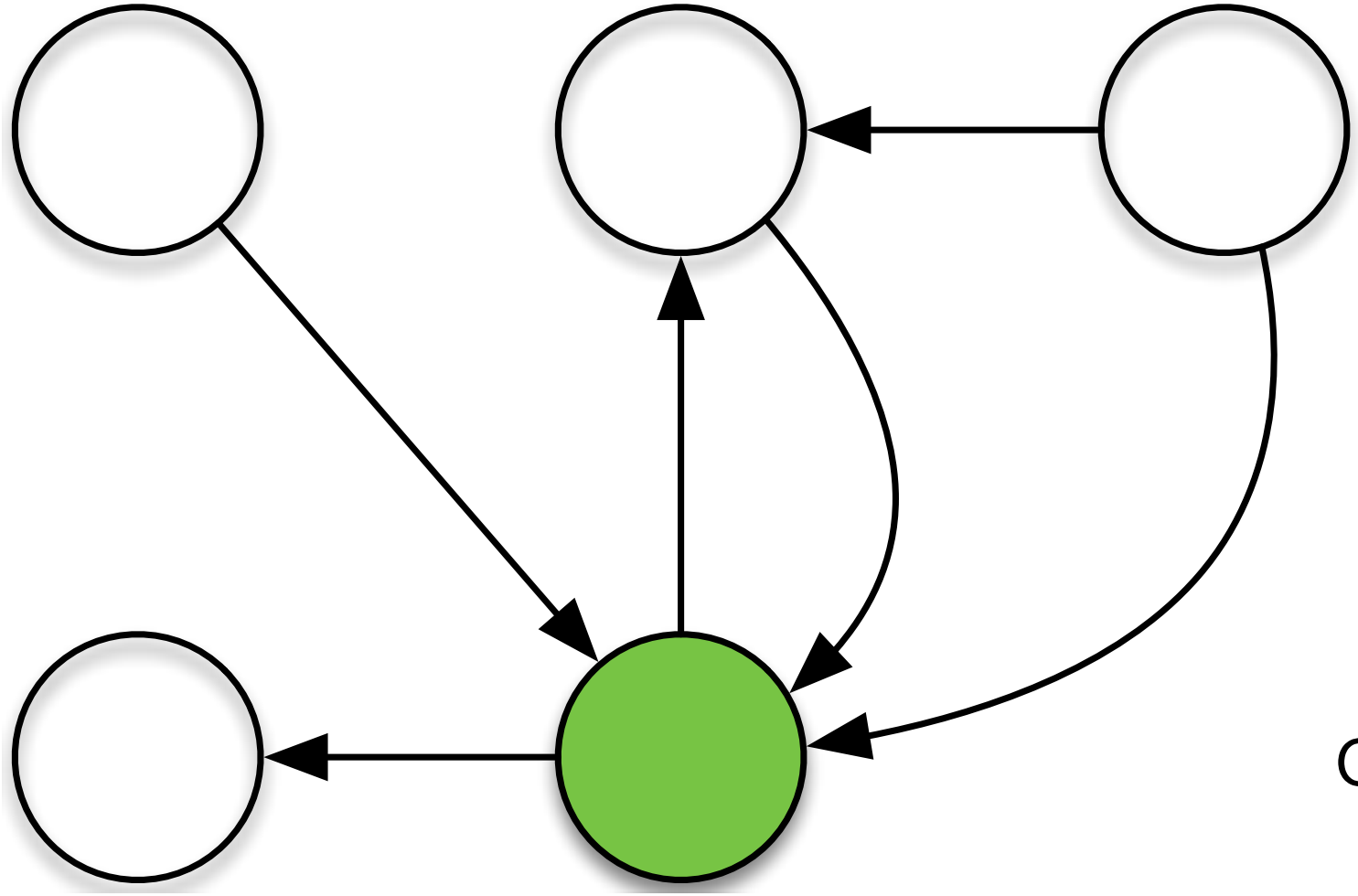
Comparison

Comparison	Winner
Faster to test if (x, y) is in graph?	adjacency matrices
Faster to find the degree of a vertex?	adjacency lists
Less memory on small graphs?	adjacency lists $(m + n)$ vs. (n^2)
Less memory on big graphs?	adjacency matrices (a small win)
Edge insertion or deletion?	adjacency matrices $O(1)$ vs. $O(d)$
Faster to traverse the graph?	adjacency lists $\Theta(m + n)$ vs. $\Theta(n^2)$
Better for most problems?	adjacency lists

Edge Concepts

Given two vertices u, v.

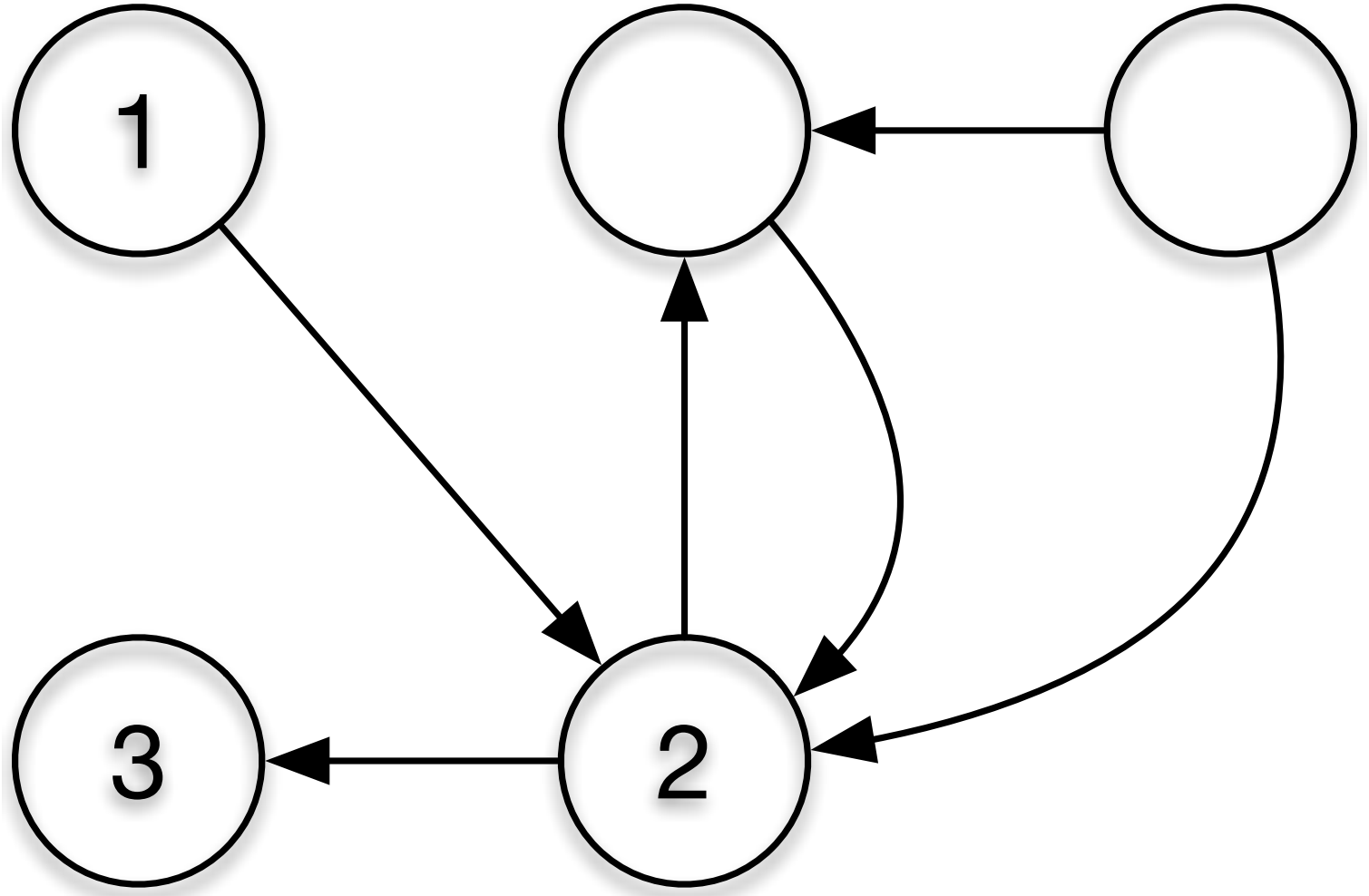
- If there is an edge from u to v, then it is incident from, or leaves u, and is incident to, or enters v. Here, v is adjacent to u.
- The number of edges entering a vertex is the vertex's in-degree, the number of edges leaving, its out-degree.
- The degree of the vertex is the sum of its in-degree and out-degree.
- For undirected graphs, the degree is the number of edges attached to the vertex, and adjacency is symmetric.



In-degree 3
Out-degree 2
Degree 5

Paths

- A path of length k from a vertex i to a vertex j — in a given graph — is a sequence of vertices $\langle v_0, \ldots, v_k \rangle$ such that $v_0 = i$ and $v_k = j$ and (v_{i-1}, v_i) is an edge, for all $i = 1, \ldots, k$.
- The path is said to contain vertices $v_0, \ldots v_k$, and edges $(v_0, v_1), \ldots (v_{k-1}, v_k)$.
- If a path exists between i and j, then j is reachable from i.
- A path is simple if all of its vertices are distinct.
- $\langle v_i, \ldots, v_j \rangle$ is a subpath of $\langle v_0, \ldots, v_k \rangle$ iff $0 \leq i \leq j \leq k$.

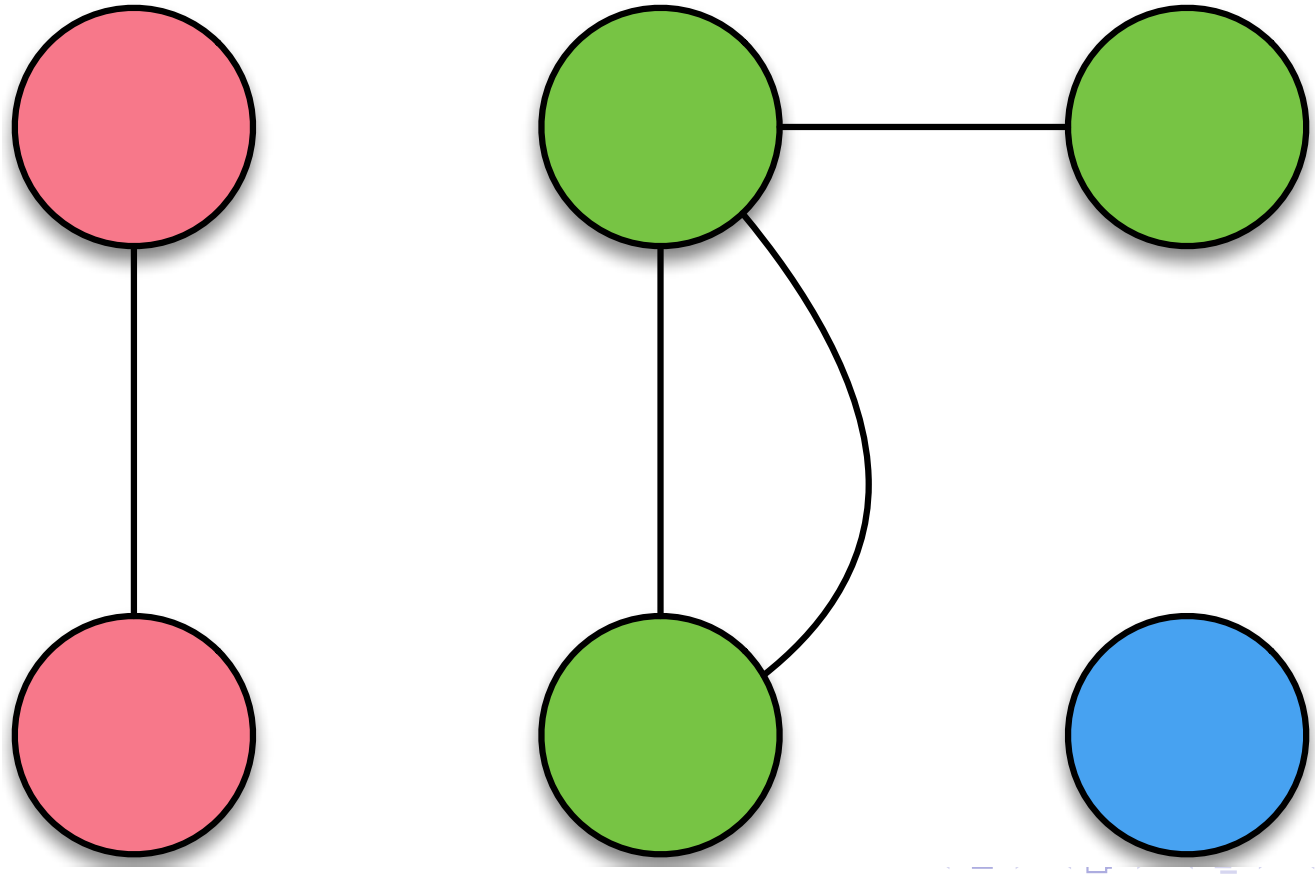

1
3
2

Cycles

- A path is a cycle if $v_0 = v_k$ and it has at least one edge.
- A self-loop is a cycle of length 1.
- It is a simple cycle if all vertices $v_1, \ldots, v_k$ are distinct. Note that a simple cycle is the smallest possible cycle.
- A graph with no cycles is acyclic.

Connectedness

- An undirected graph is connected there is a path from every vertex to every other vertex.
- Connected components are (intuitively) those vertices that are reachable from one another.



Formalisation of Connectedness

- A binary relation on a set X is a set $R \subseteq X \times X$.
  - The "less than" relation is a binary relation on the real numbers, i.e., $<\subset \mathbb{R} \times \mathbb{R}$. Thus, $(4,5) \in <$ but $(5,4)$ and $(5,5) \notin <$.
- A relation R is an equivalence relation if, for all $x, y, z \in X$, the relation satisfies

reflexivity: $(x,x) \in R$
symmetry: if $(x,y) \in R$, then $(y,x) \in R$
transitivity: if $(x,y) and (y,z) \in R$, then $(x,z) \in R$.

- Example: "has the same birthday as". Non-example: $\leq$.

Formalisation of Connectedness

- For an $x \in X$, its equivalence class is those elements $y \in X$ which satisfies the equivalence relation.

$$[x] = \{y \in X | (x, y) \in R\}$$

- Example: All people whose birthday falls on the same day, for every day of the year.
- Connected components are the equivalence classes of vertices under the reachable from relation (i.e., where there is a path between them).

Connectedness in directed graphs

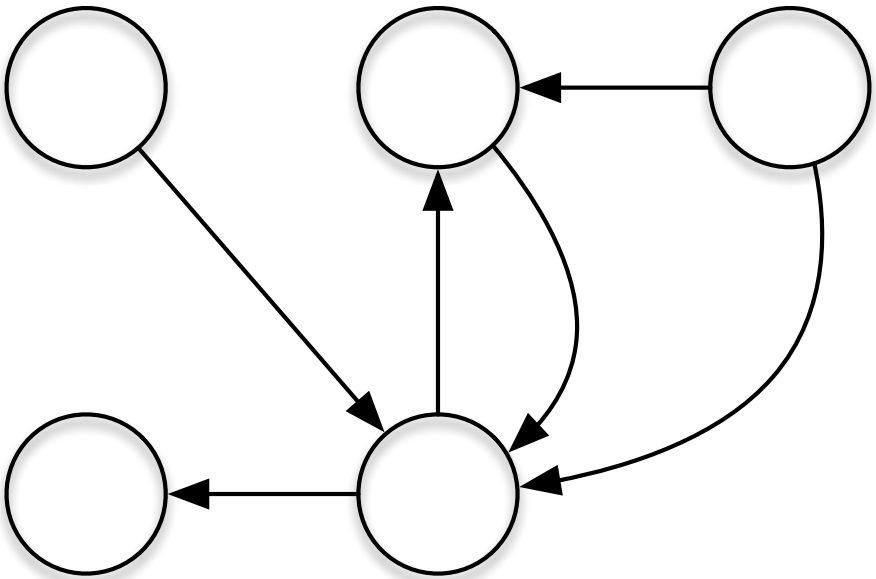
- A directed graph is strongly connected if every two vertices are reachable from each other.
- The strongly connected components are the equivalence classes of vertices that are mutually reachable.

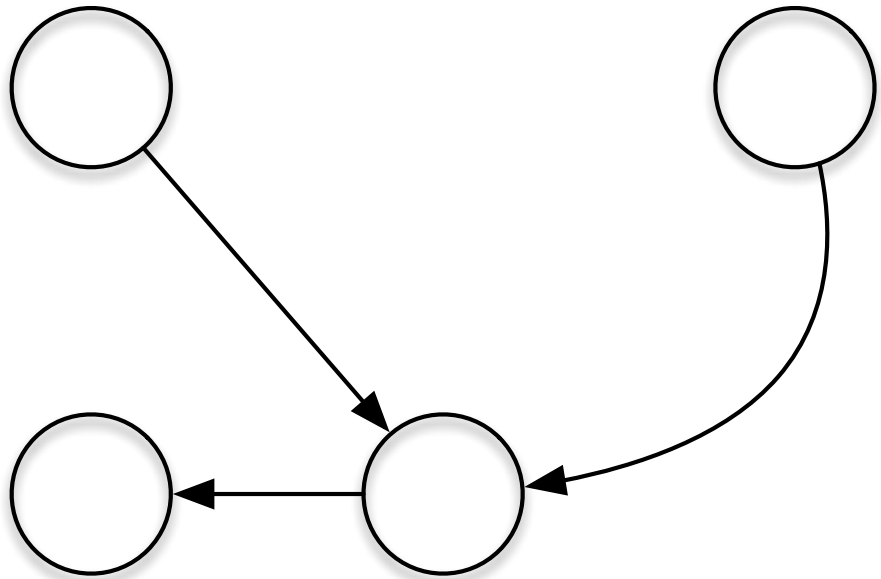
Subgraphs

- Given a graph (V, E), we can identify a subgraph $S = (V', E')$ where

$$V' \subseteq V \qquad \text{and } E' \subseteq E \cap V' \times V'$$

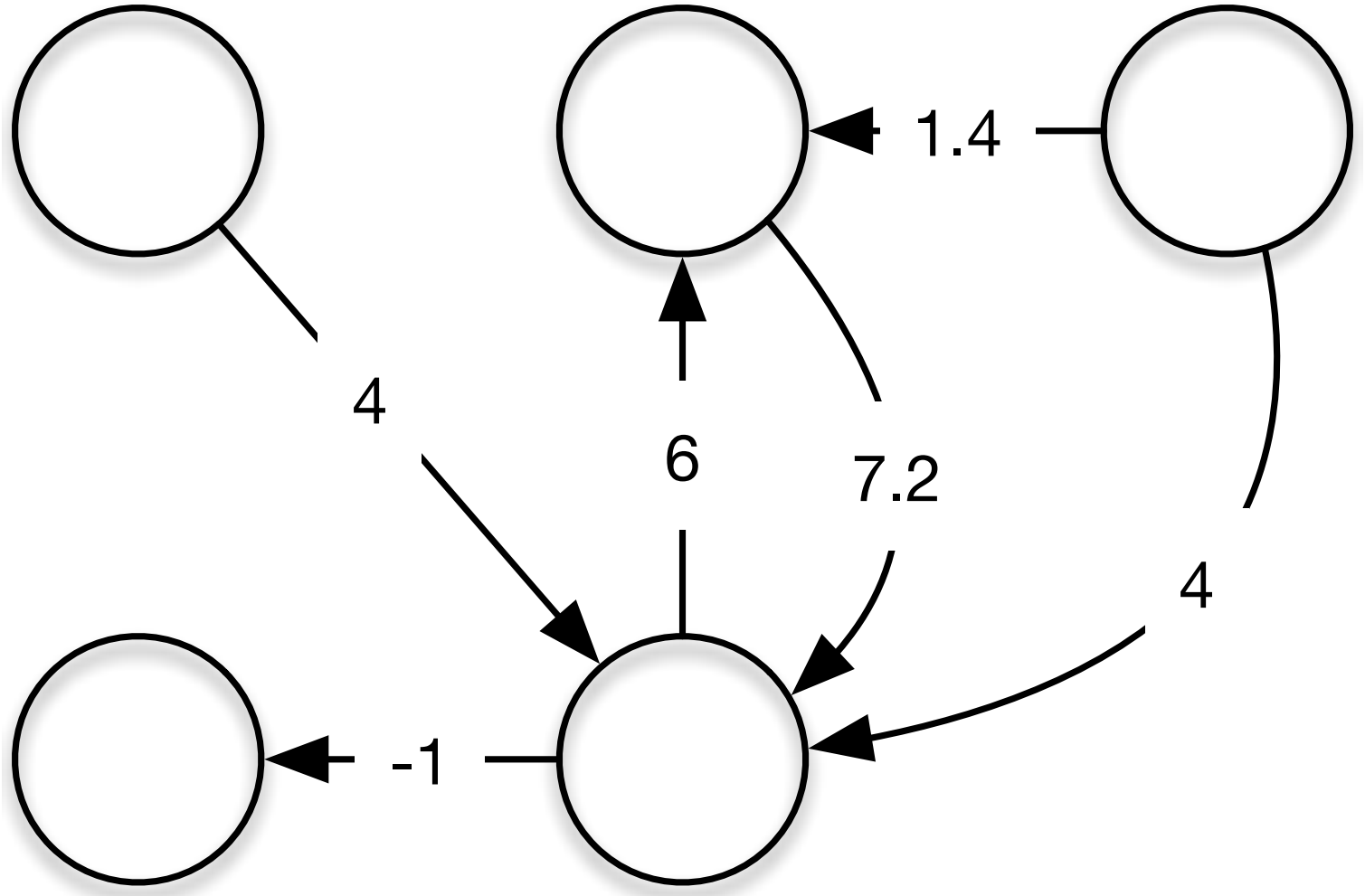
- S is the subgraph induced by V' iff $x, y \in V'$ and $(x, y) \in E$, then $(x, y) \in E'$





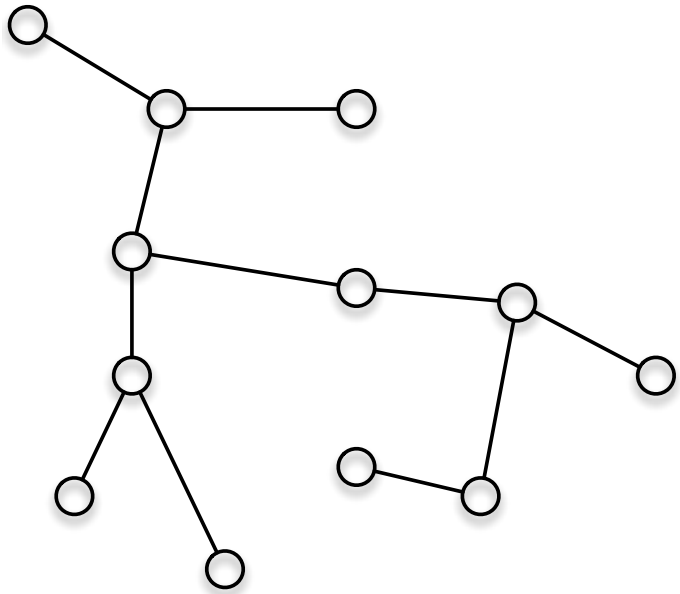
Weighted Graph

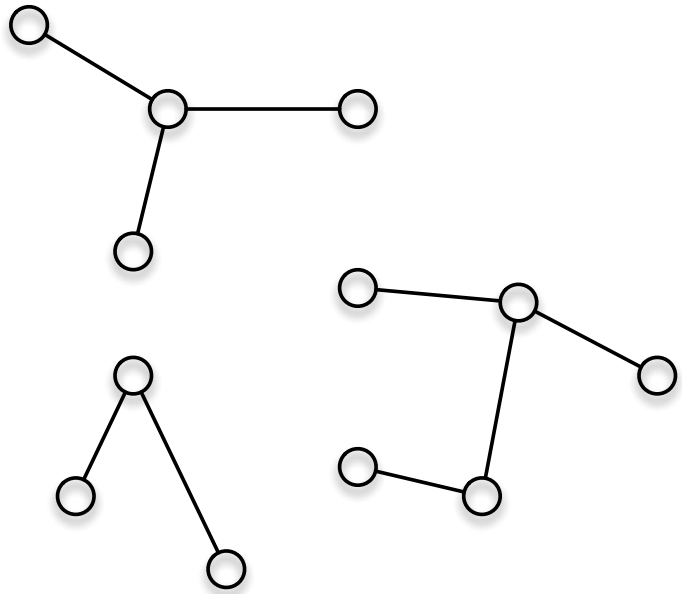
- A weighted graph has a value (typically a real or integer number) associated with it, this is the edge's weight.



Trees

- A free tree is an acyclic, connected undirected graph.
- A forest is an acyclic undirected graph.





Trees

- A rooted tree is a free tree with a single distinguished vertex called the root.
- A node v on the path between the root and node w is an ancestor of w, and w is a descendant of v. Note, if $v \neq w$, then we have a proper ancestor or descendant.
- If (v, w) is an edge in the tree, with v the ancestor of w, then v is the parent of w, and w is v's child.
- Nodes sharing a parent are called siblings.
- Child-free nodes are leaves; non-leaf nodes are internal.
- The number of children a node has are referred to as the node's degree.

Trees

- The path length from root to node is that node's depth.
- The tree's height is the maximum depth of nodes in the tree.
- A tree is ordered if the children of each node are ordered (i.e., one can refer to the first, second, etc child of a node).

Trees

- A binary tree is a finite set of nodes and consists of either
  1. An empty set of nodes; or
  2. Three disjoint sets of nodes: a singleton set containing a root node; a binary tree called th left subtree and a binary tree called the right subtree.
- Since the sets are disjoint, each child must be in either the left, or right tree (and is called a left child or right child).
- A binary tree is a full binary tree if no nodes have a single child.
- Definition can be extended to k-ary trees, where each node has at most k children, and each child is labelled.

Free Tree Properties

If $G = (V, E)$ is an undirected graph, then the following properties are equivalent.

1. G is a free tree
2. Any two vertices in G are connected by a unique simple path
3. G is connected, but if any edge is removed from E, the resulting graph is disconnected
4. G is connected, and $|E| = |V| - 1$
5. G is acyclic, and $|E| = |V| - 1$
6. G is acyclic, but if any edge is added to E, the resulting graph contains a cycle.

- For a set X, $|X|$ is the cardinality of the set. For finite sets, this is the number of elements in the set.

Free tree properties

- We won't prove all these properties (See Cormen for details), but consider $6 \rightarrow 1$
- Suppose that G is acyclic, but that adding any edge to E creates a cycle. We must show that G is connected.
- Let u and v be arbitrary vertices in G.
- Since adding the edge creates a cycle, if we assume that u and v are not adjacent, adding such an edge results a cycle whose edges, except for (u, v) belong to G.
- Thus, there must already be a path from u to v.
- Since u, v were chosen arbitrarily, G is connected.

Where are we?

- We've introduced graphs - structures made up of nodes and edges.
- Considered how graphs can be represented.
- Examined different types of graphs (weighted, directed, undirected, trees).
- Defined a variety of concepts over graphs.
- We are now able to "speak in the language of graphs", allowing us to examine algorithms which manipulate and use graphs for various purposes.