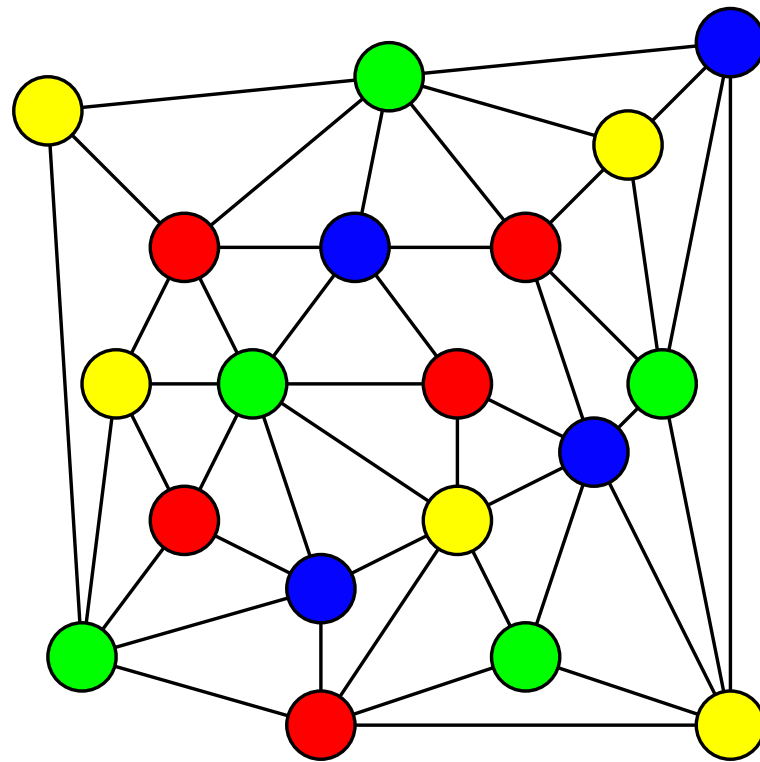


# Algorithms and Analysis

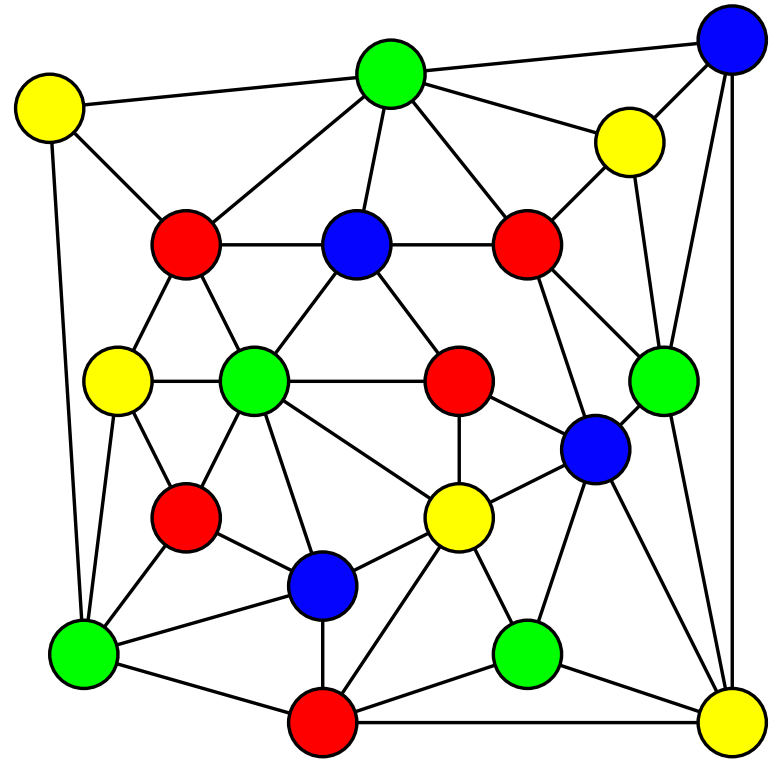
## Lesson 16: *Think Graphically*



*Graph theory, applications of graphs, graph problems*

# Outline

1. **Graph Theory**
2. Applications of Graphs
  - Geometric applications
  - Relational applications
3. Implementing Graphs
4. Graph Problems



# Motivation

- Many different problems can be described in terms of graphs
- This often reveals the true nature of the problem
- It unifies many apparently different problems
- As much is known about graph problems it often provides a pointer to the solution

# Motivation

- Many different problems can be described in terms of graphs
- This often reveals the true nature of the problem
- It unifies many apparently different problems
- As much is known about graph problems it often provides a pointer to the solution

# Motivation

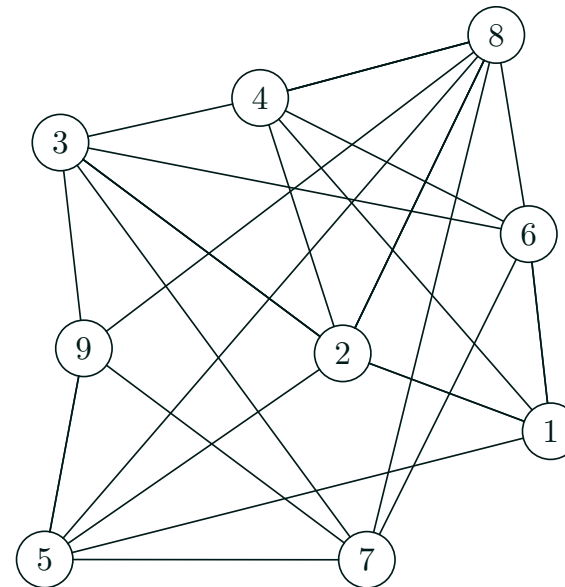
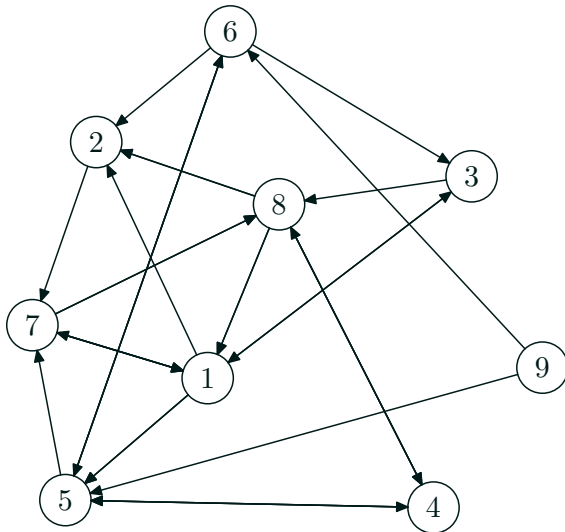
- Many different problems can be described in terms of graphs
- This often reveals the true nature of the problem
- It unifies many apparently different problems
- As much is known about graph problems it often provides a pointer to the solution

# Motivation

- Many different problems can be described in terms of graphs
- This often reveals the true nature of the problem
- It unifies many apparently different problems
- As much is known about graph problems it often provides a pointer to the solution

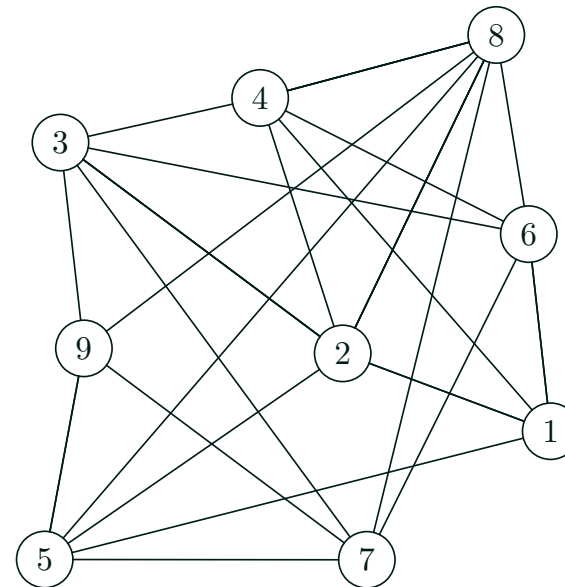
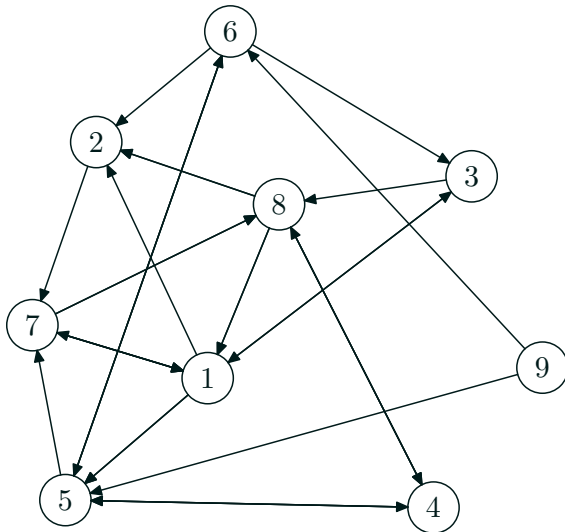
# Definition of a Graph

- A graph,  $G$ , can be described by
  - ★ A set of vertices or nodes  $\mathcal{V} = \{1, 2, 3 \dots n\}$
  - ★ A set of edges  $\mathcal{E} = \{(i, j) | \text{vertex } i \text{ is connected to vertex } j\}$
- The edges may be
  - ★ **directed**—sometimes called a **digraph**
  - ★ **undirected**



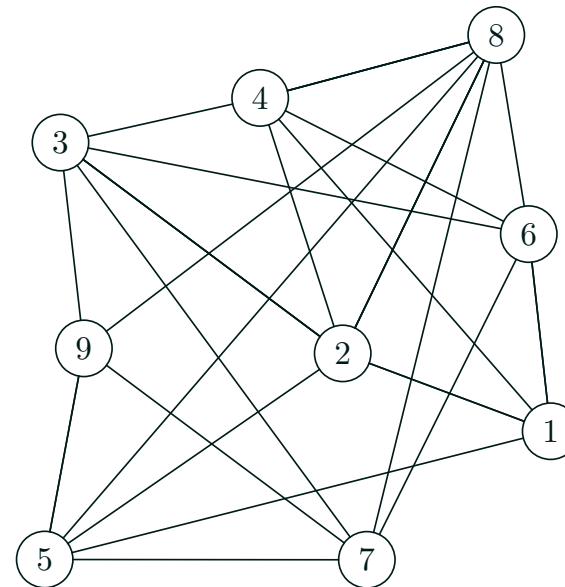
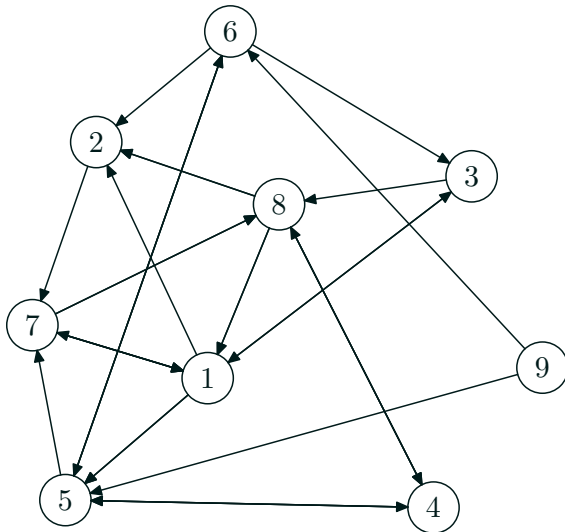
# Definition of a Graph

- A graph,  $G$ , can be described by
  - ★ A set of vertices or nodes  $\mathcal{V} = \{1, 2, 3 \dots n\}$
  - ★ A set of edges  $\mathcal{E} = \{(i, j) | \text{vertex } i \text{ is connected to vertex } j\}$
- The edges may be
  - ★ **directed**—sometimes called a **digraph**
  - ★ **undirected**



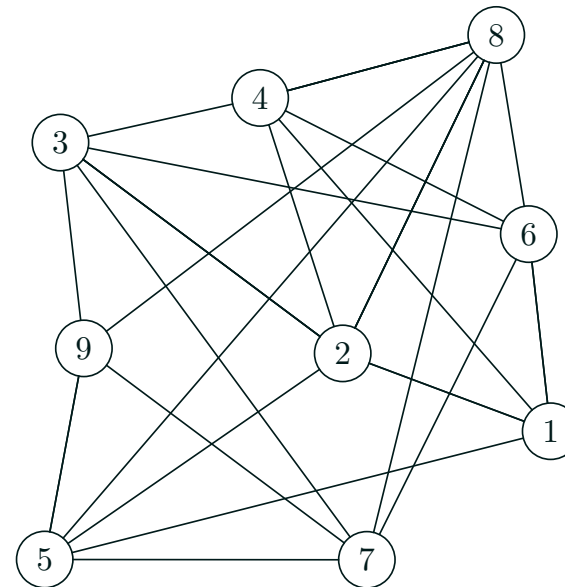
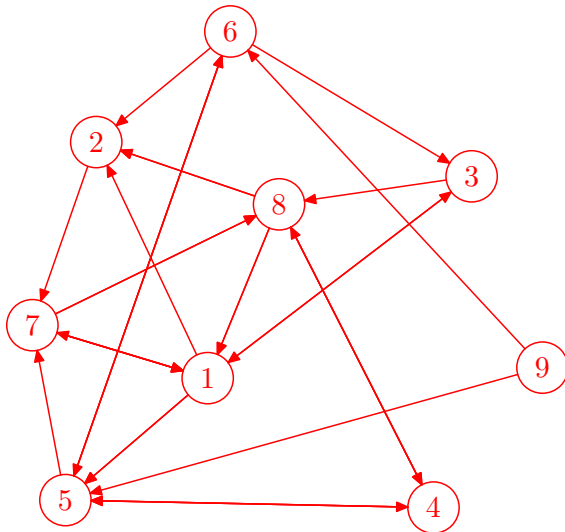
# Definition of a Graph

- A graph,  $G$ , can be described by
  - ★ A set of vertices or nodes  $\mathcal{V} = \{1, 2, 3 \dots n\}$
  - ★ A set of edges  $\mathcal{E} = \{(i, j) | \text{vertex } i \text{ is connected to vertex } j\}$
- The edges may be
  - ★ **directed**—sometimes called a **digraph**
  - ★ **undirected**



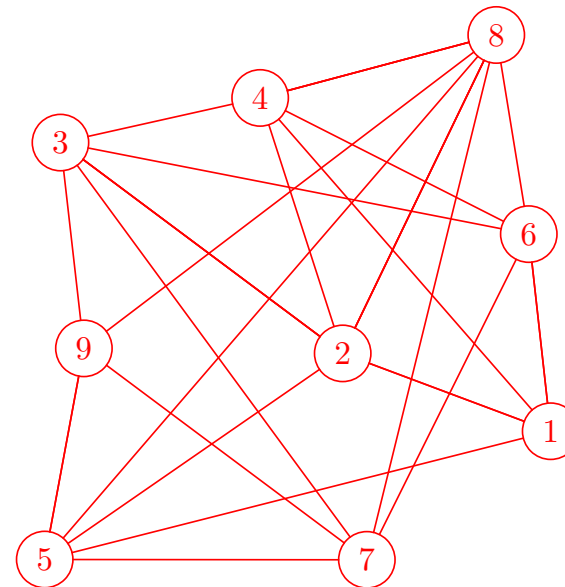
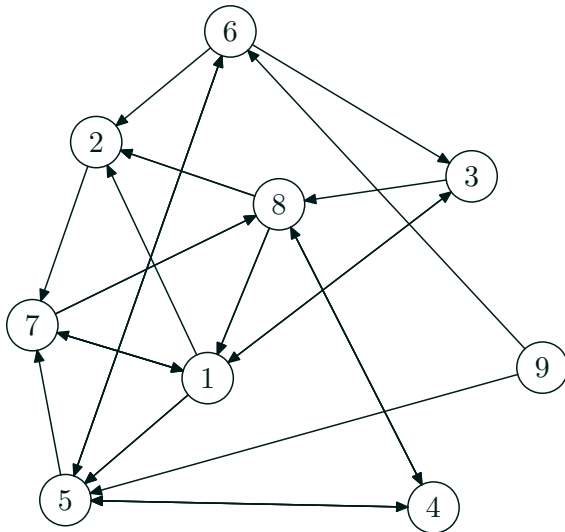
# Definition of a Graph

- A graph,  $G$ , can be described by
  - ★ A set of vertices or nodes  $\mathcal{V} = \{1, 2, 3 \dots n\}$
  - ★ A set of edges  $\mathcal{E} = \{(i, j) | \text{vertex } i \text{ is connected to vertex } j\}$
- The edges may be
  - ★ **directed**—sometimes called a **digraph**
  - ★ **undirected**



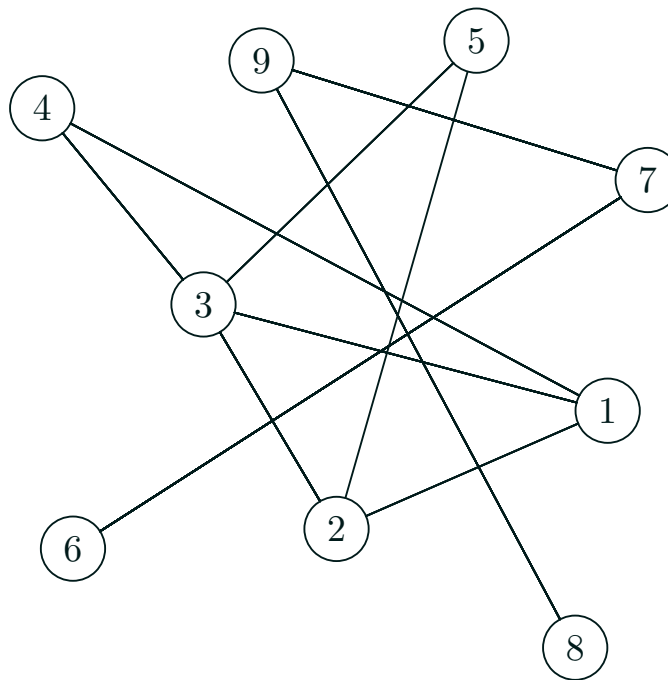
# Definition of a Graph

- A graph,  $G$ , can be described by
  - ★ A set of vertices or nodes  $\mathcal{V} = \{1, 2, 3 \dots n\}$
  - ★ A set of edges  $\mathcal{E} = \{(i, j) | \text{vertex } i \text{ is connected to vertex } j\}$
- The edges may be
  - ★ **directed**—sometimes called a **digraph**
  - ★ **undirected**



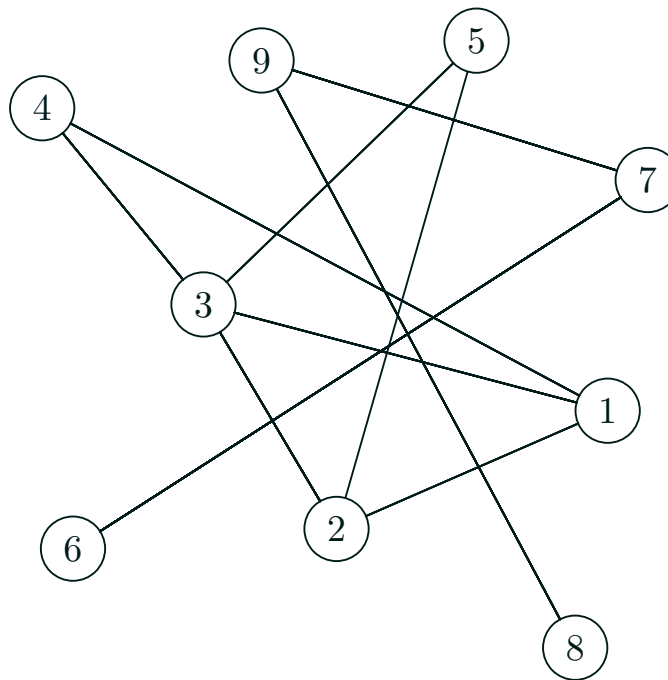
# Connected and Unconnected Graphs

- A graph is **connected** if you can get from one node to any other along a series of edges
- Otherwise it is **disconnected**



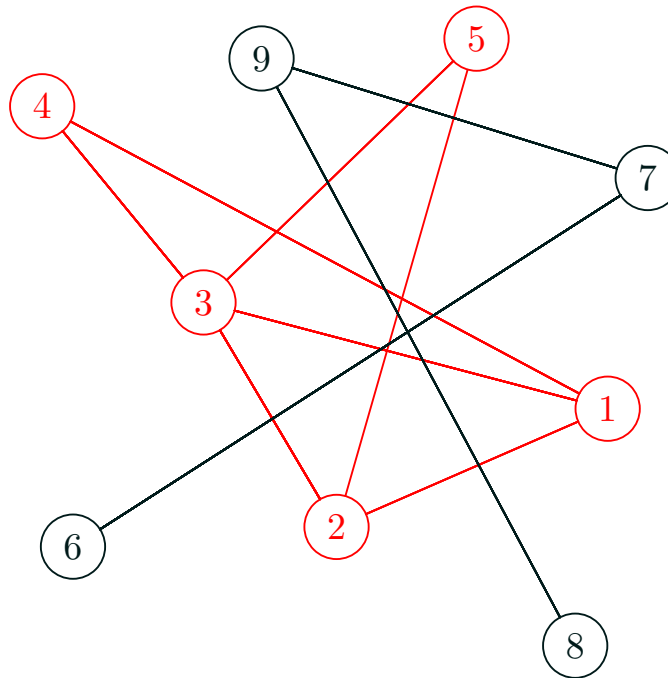
# Connected and Unconnected Graphs

- A graph is **connected** if you can get from one node to any other along a series of edges
- Otherwise it is **disconnected**



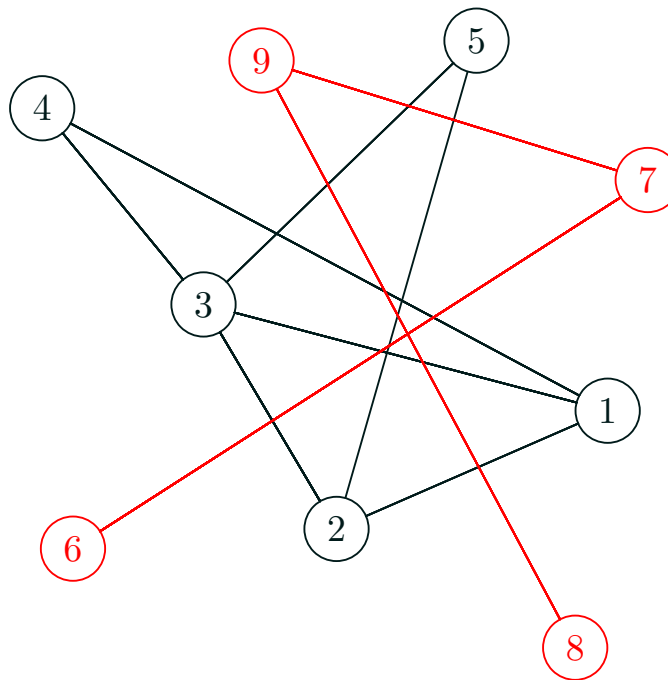
# Connected and Unconnected Graphs

- A graph is **connected** if you can get from one node to any other along a series of edges
- Otherwise it is **disconnected**



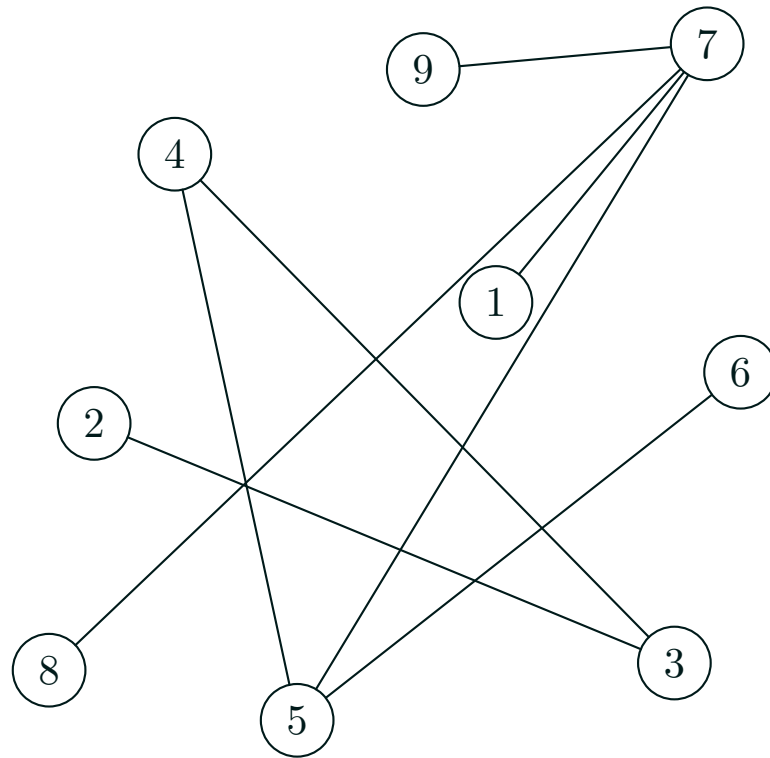
# Connected and Unconnected Graphs

- A graph is **connected** if you can get from one node to any other along a series of edges
- Otherwise it is **disconnected**



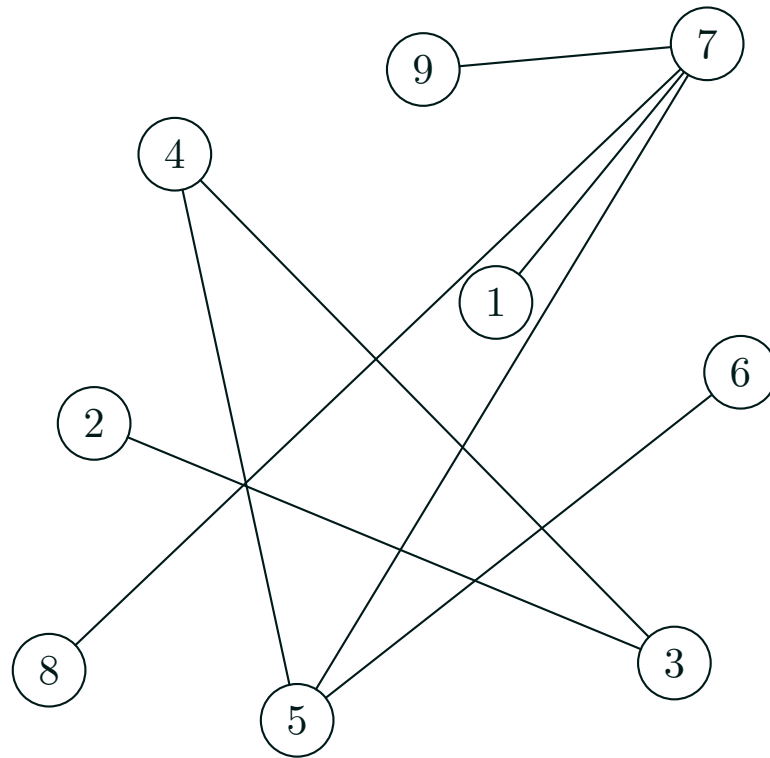
# Trees

- A tree is a connected graphs with no cycles
- A tree will have  $n - 1$  edges



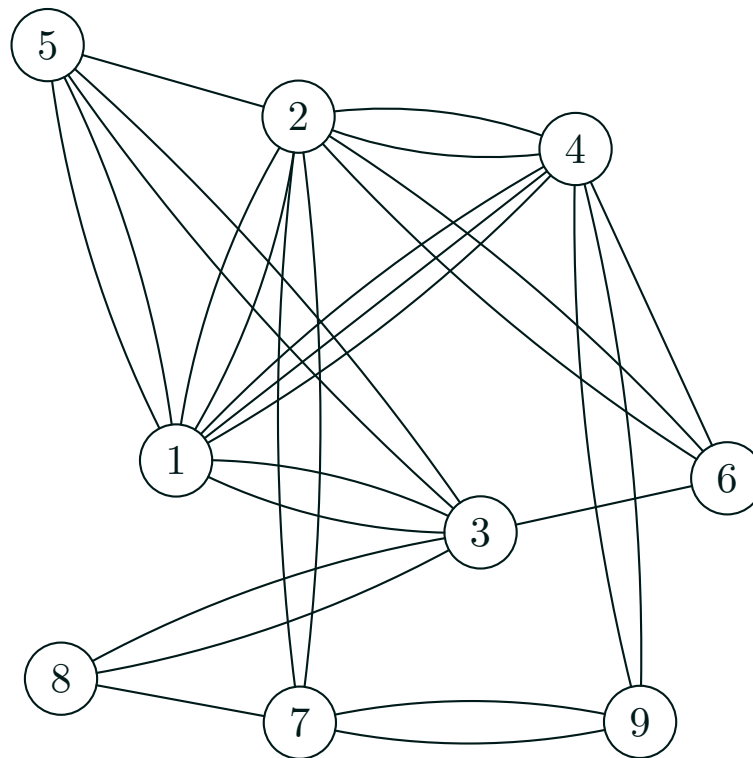
# Trees

- A tree is a connected graphs with no cycles
- A tree will have  $n - 1$  edges



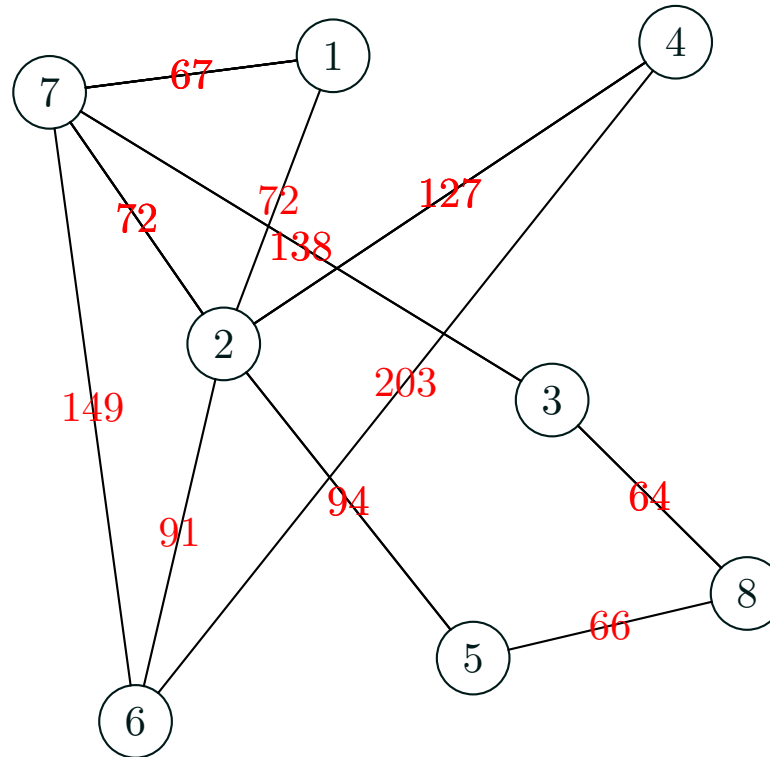
# Multigraphs

- If the collection of edges is a *multiset* then we obtain a **multigraphs** where more than one edge is allowed between pairs of vertices



# Weighted Graphs

- If we assign a number to an edge we obtain a **weighted graph**



# Networks

- Sometimes we add more information to the graph
- E.g. attributes to the nodes or edges
- Graphs with many attributes are often referred to as **networks**

# Networks

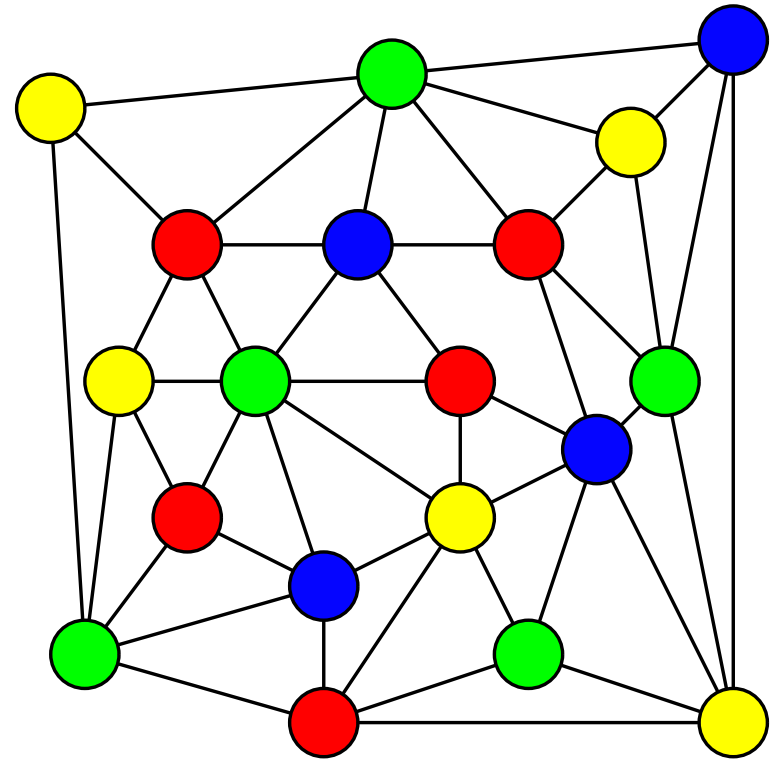
- Sometimes we add more information to the graph
- E.g. attributes to the nodes or edges
- Graphs with many attributes are often referred to as **networks**

# Networks

- Sometimes we add more information to the graph
- E.g. attributes to the nodes or edges
- Graphs with many attributes are often referred to as **networks**

# Outline

1. Graph Theory
2. **Applications of Graphs**
  - Geometric applications
  - Relational applications
3. Implementing Graphs
4. Graph Problems



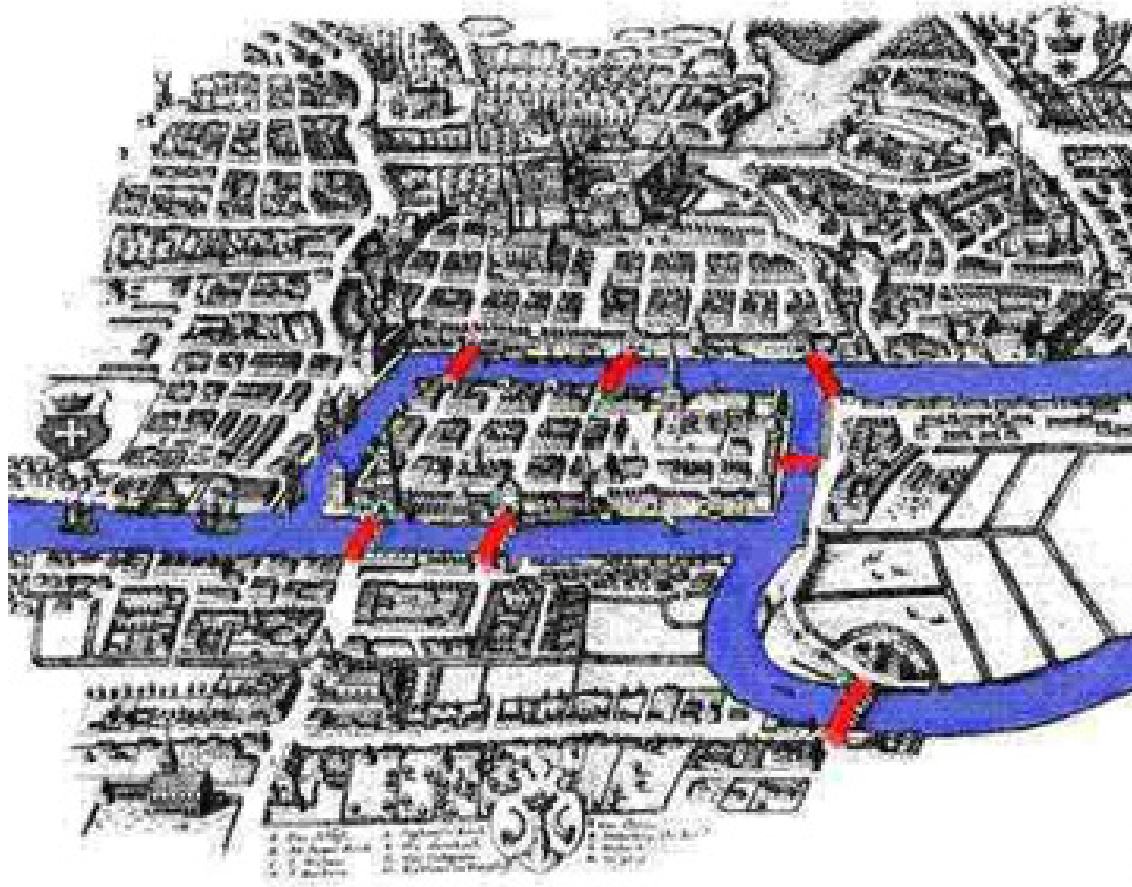
# Bridges of Königsberg

Is there a tour around Königsberg going over every bridge once?



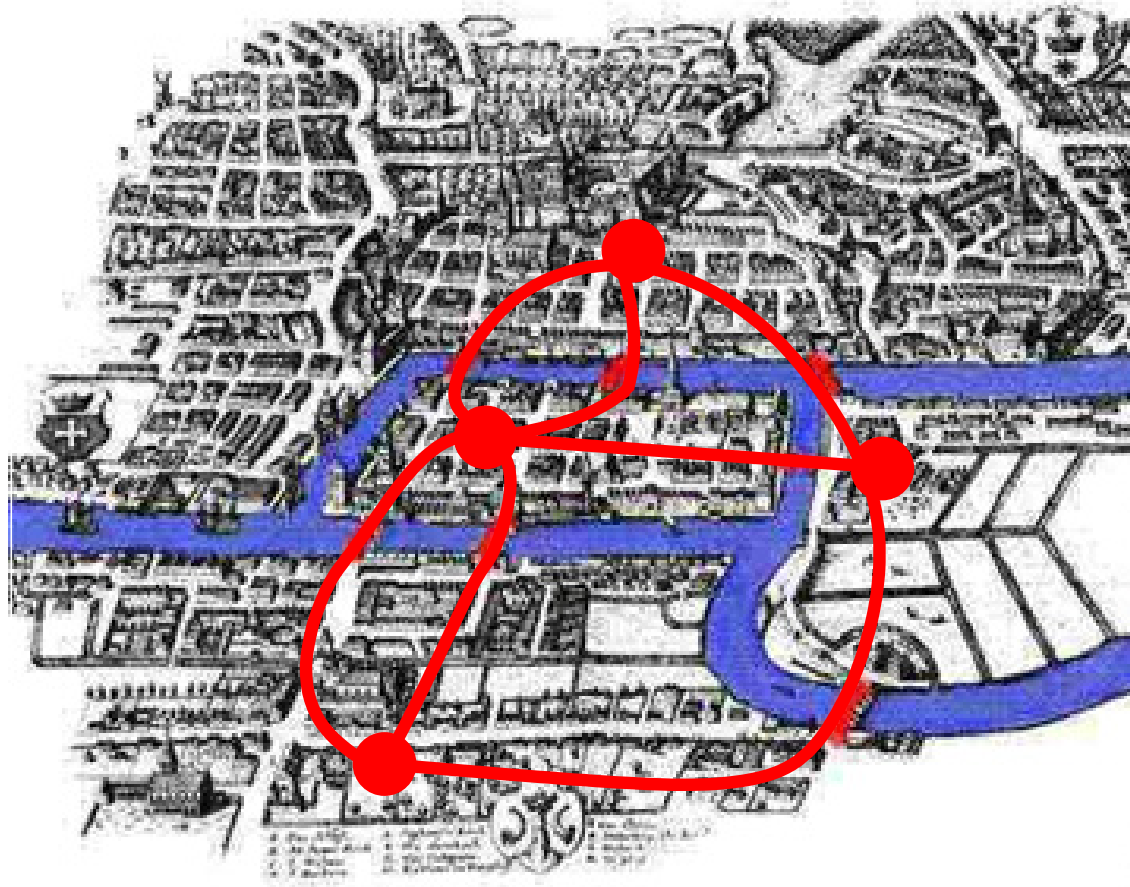
# Bridges of Königsberg

Is there a tour around Königsberg going over every bridge once?



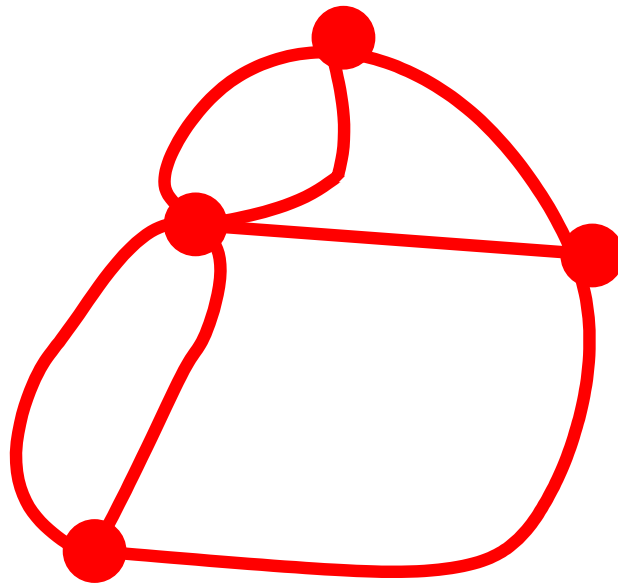
# Bridges of Königsberg

Is there a tour around Königsberg going over every bridge once?



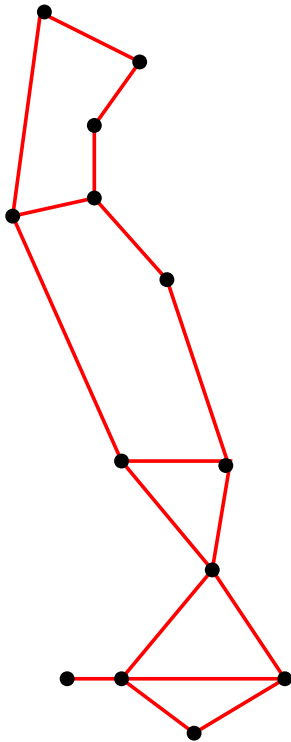
# Bridges of Königsberg

Is there a tour around Königsberg going over every bridge once?



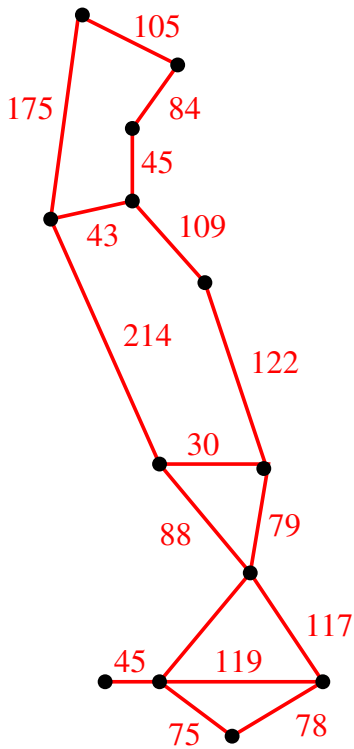
In 1736 Euler published a paper answering this question and founding graph theory

# Representing Distances



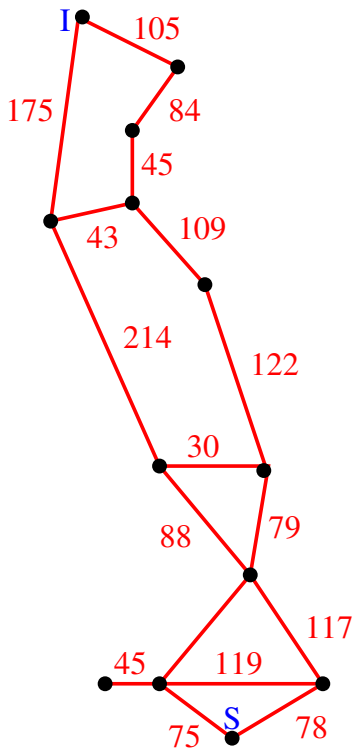
- Consider some graph
- With weights representing the distance between nodes
- What is the shortest distance between  $S$  and  $I$ ?

# Representing Distances



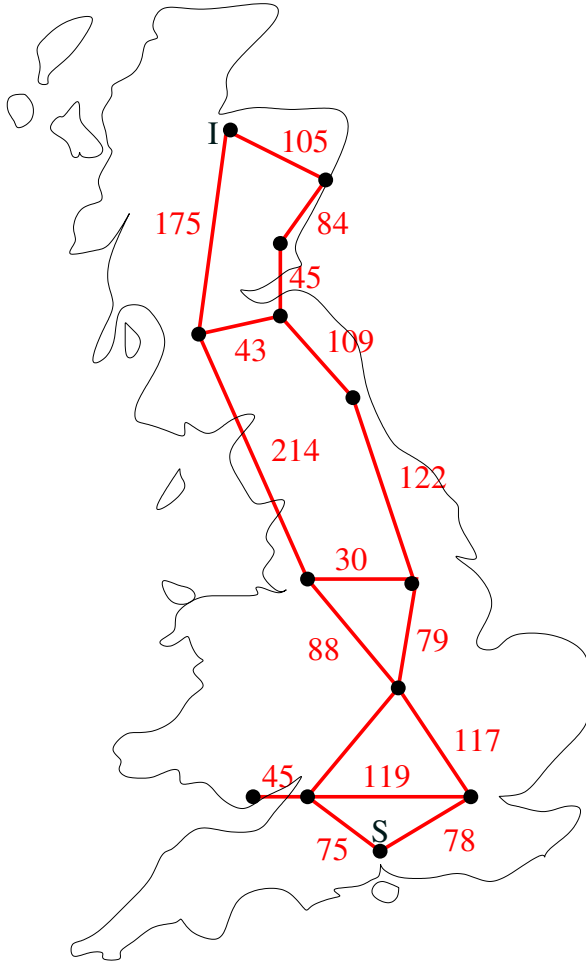
- Consider some graph
- With weights representing the distance between nodes
- What is the shortest distance between  $S$  and  $I$ ?

# Representing Distances



- Consider some graph
- With weights representing the distance between nodes
- What is the shortest distance between  $S$  and  $I$ ?

# Representing Distances



# Other Applications

- We could take the weights to represent the time taken to travel between nodes
- In a computer network the weights might represent the bandwidth
- In a representation of a transport system the weights might represent the carrying capacity of the traffic on a road
- Graphs can be used to represent other kinds of relationships
- E.g. We could create a digraph of links between web pages

# Other Applications

- We could take the weights to represent the time taken to travel between nodes
- In a computer network the weights might represent the bandwidth
- In a representation of a transport system the weights might represent the carrying capacity of the traffic on a road
- Graphs can be used to represent other kinds of relationships
- E.g. We could create a digraph of links between web pages

# Other Applications

- We could take the weights to represent the time taken to travel between nodes
- In a computer network the weights might represent the bandwidth
- In a representation of a transport system the weights might represent the carrying capacity of the traffic on a road
- Graphs can be used to represent other kinds of relationships
- E.g. We could create a digraph of links between web pages

# Other Applications

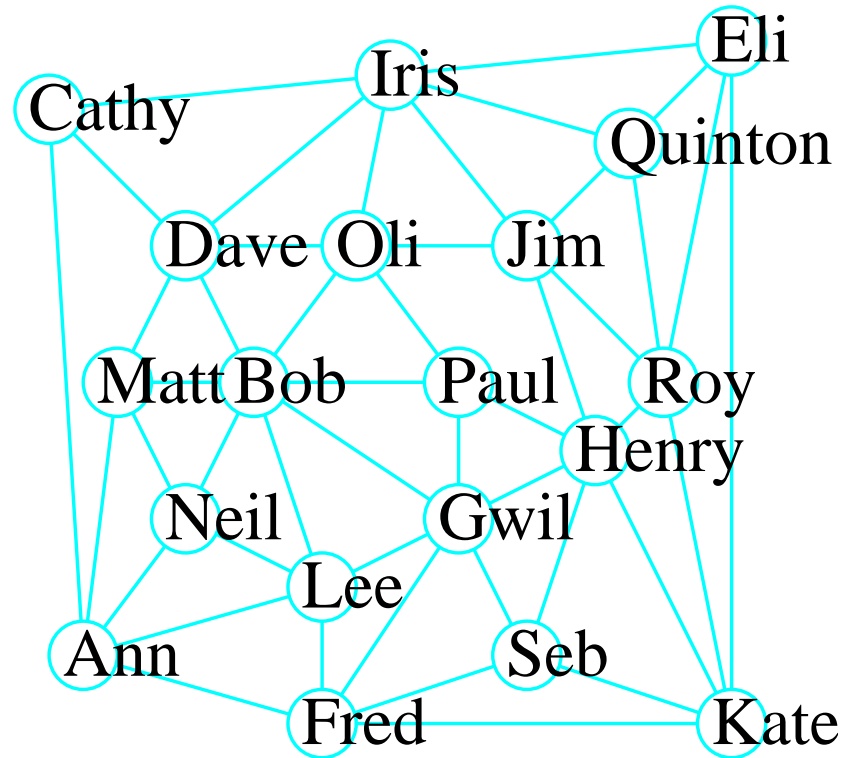
- We could take the weights to represent the time taken to travel between nodes
- In a computer network the weights might represent the bandwidth
- In a representation of a transport system the weights might represent the carrying capacity of the traffic on a road
- Graphs can be used to represent other kinds of relationships
- E.g. We could create a digraph of links between web pages

# Other Applications

- We could take the weights to represent the time taken to travel between nodes
- In a computer network the weights might represent the bandwidth
- In a representation of a transport system the weights might represent the carrying capacity of the traffic on a road
- Graphs can be used to represent other kinds of relationships
- E.g. We could create a digraph of links between web pages

# Christmas Card Problem

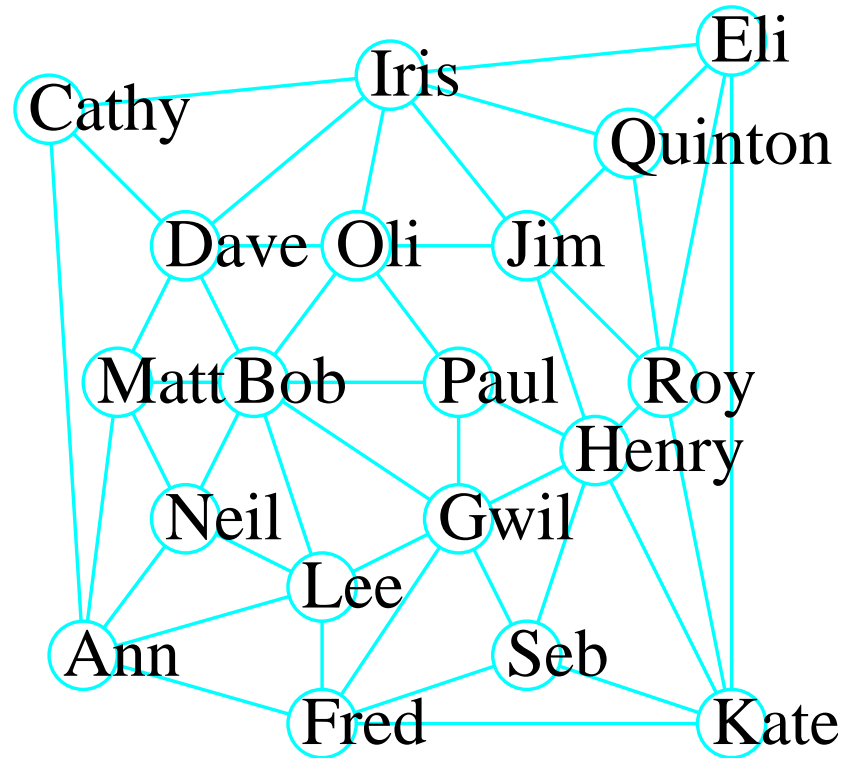
- I have four types of Christmas cards
- Some of my friends know each other



- I don't want to send friends that know each other the same card

# Christmas Card Problem

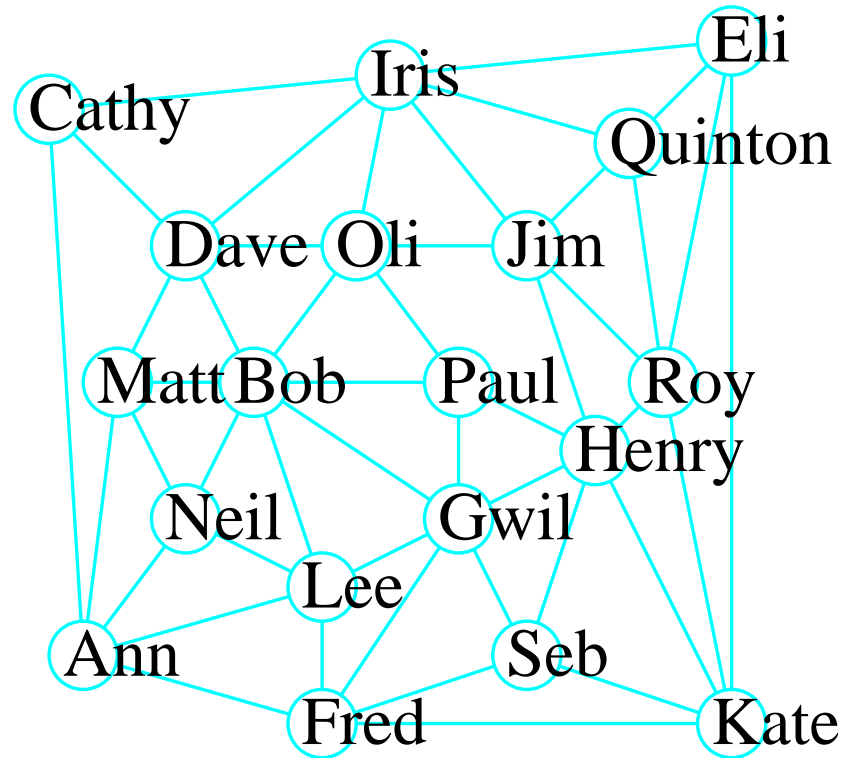
- I have four types of Christmas cards
- Some of my friends know each other



- I don't want to send friends that know each other the same card

# Christmas Card Problem

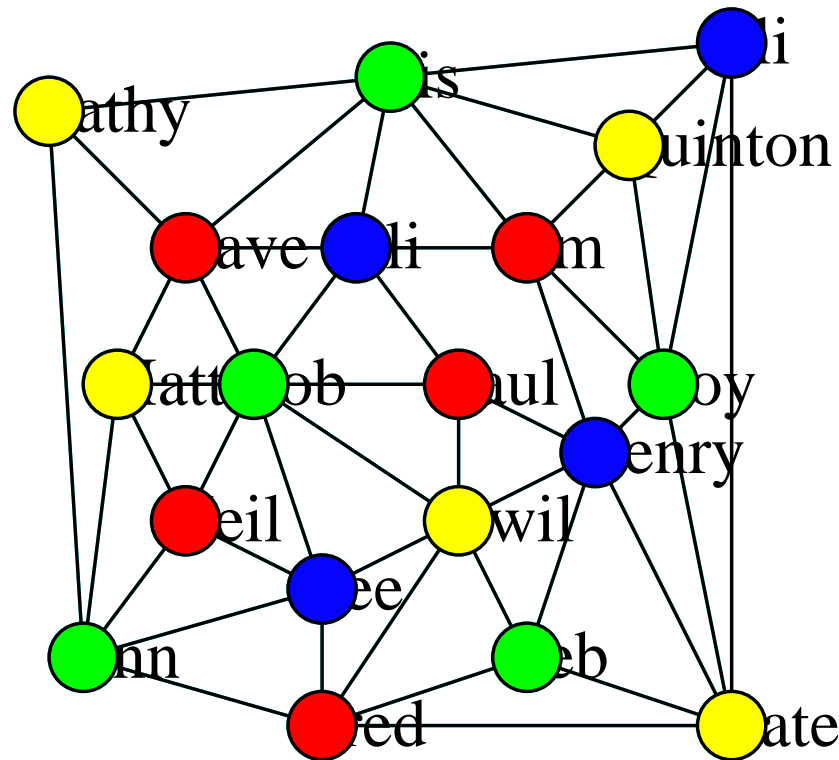
- I have four types of Christmas cards
- Some of my friends know each other



- I don't want to send friends that know each other the same card

# Christmas Card Problem

- I have four types of Christmas cards
- Some of my friends know each other



- I don't want to send friends that know each other the same card

# A Real World Problem

- A food company used different colour bags for each of its products
- To save money they reduced the stock of bags to 25
- They wanted to know what items to put in what bags so that as few customers as possible would have items with the same colour bags
- This can again be reduced to a graph colouring problem
  - ★ Each node represents an item
  - ★ The edges were weighted by the number of customers that took both items
  - ★ The aim was to colour the nodes with 25 colours to minimise the weights where the edges shared the same colour

# A Real World Problem

- A food company used different colour bags for each of its products
- To save money they reduced the stock of bags to 25
- They wanted to know what items to put in what bags so that as few customers as possible would have items with the same colour bags
- This can again be reduced to a graph colouring problem
  - ★ Each node represents an item
  - ★ The edges were weighted by the number of customers that took both items
  - ★ The aim was to colour the nodes with 25 colours to minimise the weights where the edges shared the same colour

# A Real World Problem

- A food company used different colour bags for each of its products
- To save money they reduced the stock of bags to 25
- They wanted to know what items to put in what bags so that as few customers as possible would have items with the same colour bags
- This can again be reduced to a graph colouring problem
  - ★ Each node represents an item
  - ★ The edges were weighted by the number of customers that took both items
  - ★ The aim was to colour the nodes with 25 colours to minimise the weights where the edges shared the same colour

# A Real World Problem

- A food company used different colour bags for each of its products
- To save money they reduced the stock of bags to 25
- They wanted to know what items to put in what bags so that as few customers as possible would have items with the same colour bags
- This can again be reduced to a graph colouring problem
  - ★ Each node represents an item
  - ★ The edges were weighted by the number of customers that took both items
  - ★ The aim was to colour the nodes with 25 colours to minimise the weights where the edges shared the same colour

# A Real World Problem

- A food company used different colour bags for each of its products
- To save money they reduced the stock of bags to 25
- They wanted to know what items to put in what bags so that as few customers as possible would have items with the same colour bags
- This can again be reduced to a graph colouring problem
  - ★ Each node represents an item
  - ★ The edges were weighted by the number of customers that took both items
  - ★ The aim was to colour the nodes with 25 colours to minimise the weights where the edges shared the same colour

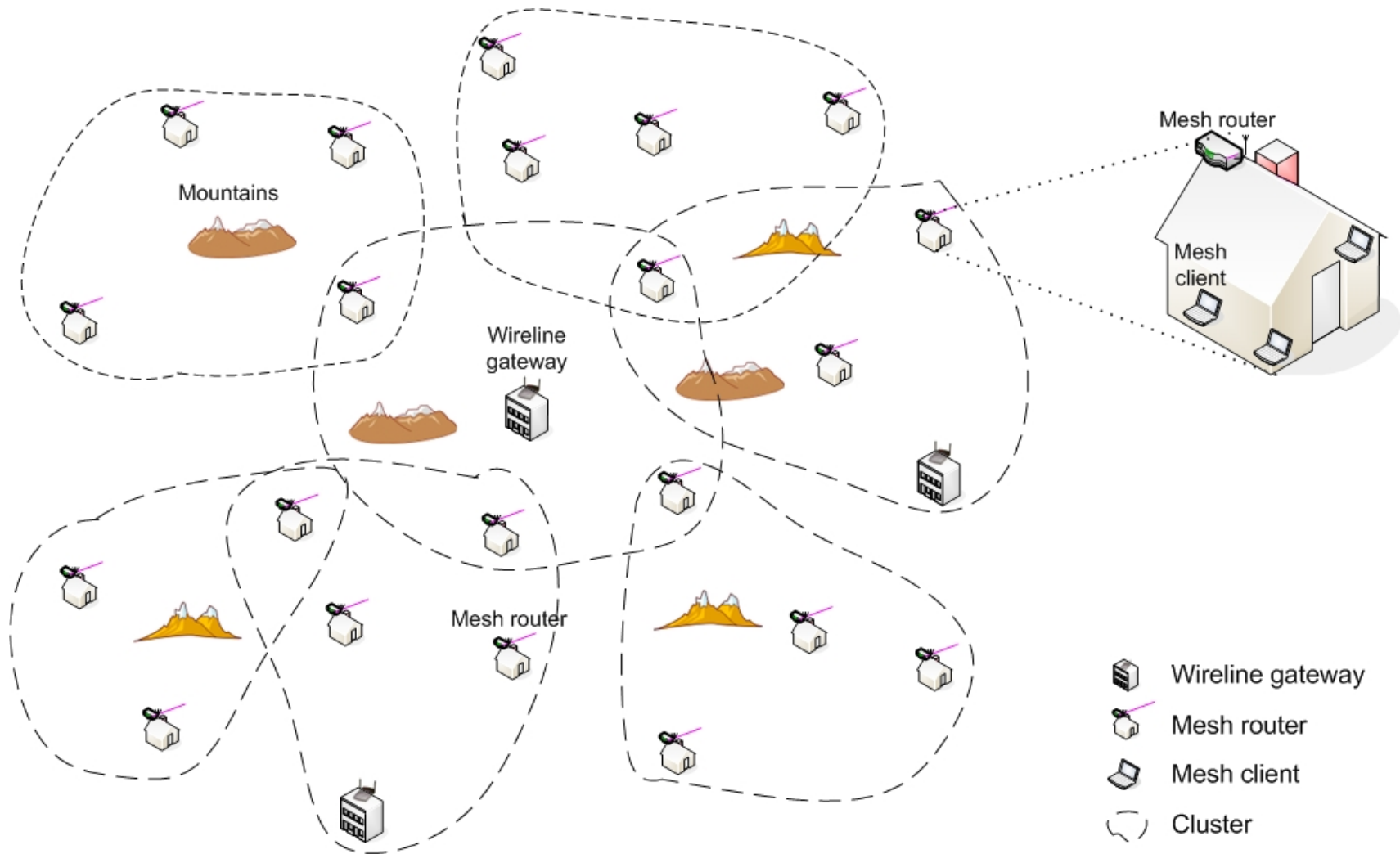
# A Real World Problem

- A food company used different colour bags for each of its products
- To save money they reduced the stock of bags to 25
- They wanted to know what items to put in what bags so that as few customers as possible would have items with the same colour bags
- This can again be reduced to a graph colouring problem
  - ★ Each node represents an item
  - ★ The edges were weighted by the number of customers that took both items
  - ★ The aim was to colour the nodes with 25 colours to minimise the weights where the edges shared the same colour

# A Real World Problem

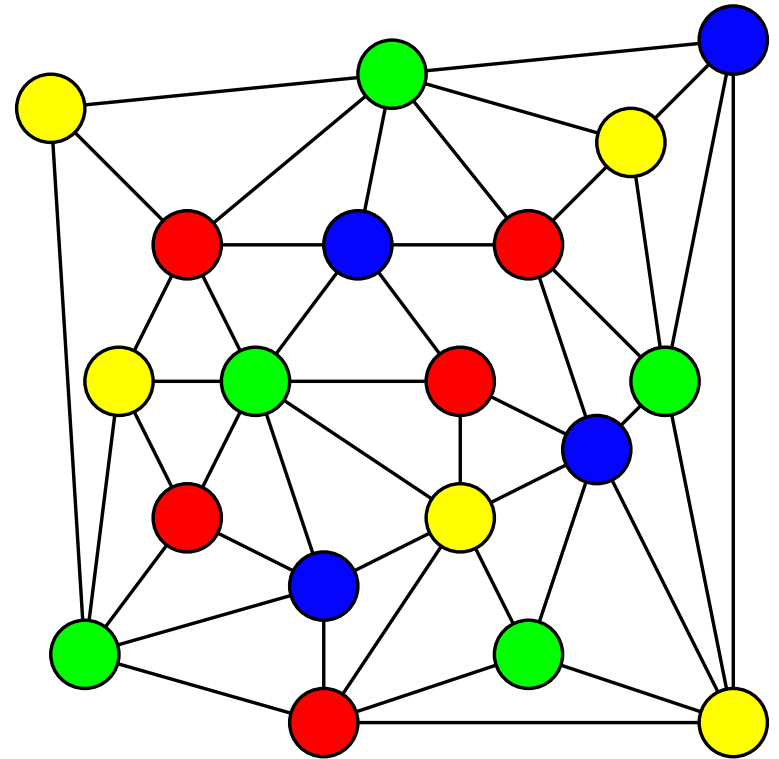
- A food company used different colour bags for each of its products
- To save money they reduced the stock of bags to 25
- They wanted to know what items to put in what bags so that as few customers as possible would have items with the same colour bags
- This can again be reduced to a graph colouring problem
  - ★ Each node represents an item
  - ★ The edges were weighted by the number of customers that took both items
  - ★ The aim was to colour the nodes with 25 colours to minimise the weights where the edges shared the same colour

# Frequency Assignment Problem



# Outline

1. Graph Theory
2. Applications of Graphs
  - Geometric applications
  - Relational applications
3. **Implementing Graphs**
4. Graph Problems



# Representations

- There is no single way to represent graphs
- The best representation depends on the graph
- Some books describe a *Graph ADT*—graphs are too varied for this to be very useful
- An important issue in representing a graph is how to store the edge information

# Representations

- There is no single way to represent graphs
- The best representation depends on the graph
- Some books describe a *Graph ADT*—graphs are too varied for this to be very useful
- An important issue in representing a graph is how to store the edge information

# Representations

- There is no single way to represent graphs
- The best representation depends on the graph
- Some books describe a *Graph ADT*—graphs are too varied for this to be very useful
- An important issue in representing a graph is how to store the edge information

# Representations

- There is no single way to represent graphs
- The best representation depends on the graph
- Some books describe a *Graph ADT*—graphs are too varied for this to be very useful
- An important issue in representing a graph is how to store the edge information

# Adjacency Matrices

- One representation of a graph  $G = (\mathcal{V}, \mathcal{E})$  is in term of an  $n \times n$  **adjacency matrix**  $\mathbf{A}$  with elements

$$A_{ij} = \begin{cases} 1 & \text{if } (i, j) \in \mathcal{E} \\ 0 & \text{if } (i, j) \notin \mathcal{E} \end{cases}$$

where  $n = |\mathcal{V}|$

- For undirected graphs  $\mathbf{A}$  is a symmetric matrix, i.e.  $\mathbf{A} = \mathbf{A}^T$
- For weighted graphs we often store the **connectivity matrix** or **cost-adjacency matrix**,  $\mathbf{C}$ , where

$$C_{ij} = \begin{cases} w_{ij} & \text{if } (i, j) \in \mathcal{E} \\ 0 & \text{if } (i, j) \notin \mathcal{E} \end{cases}$$

# Adjacency Matrices

- One representation of a graph  $G = (\mathcal{V}, \mathcal{E})$  is in term of an  $n \times n$  **adjacency matrix**  $\mathbf{A}$  with elements

$$A_{ij} = \begin{cases} 1 & \text{if } (i, j) \in \mathcal{E} \\ 0 & \text{if } (i, j) \notin \mathcal{E} \end{cases}$$

where  $n = |\mathcal{V}|$

- For undirected graphs  $\mathbf{A}$  is a symmetric matrix, i.e.  $\mathbf{A} = \mathbf{A}^T$
- For weighted graphs we often store the **connectivity matrix** or **cost-adjacency matrix**,  $\mathbf{C}$ , where

$$C_{ij} = \begin{cases} w_{ij} & \text{if } (i, j) \in \mathcal{E} \\ 0 & \text{if } (i, j) \notin \mathcal{E} \end{cases}$$

# Adjacency Matrices

- One representation of a graph  $G = (\mathcal{V}, \mathcal{E})$  is in term of an  $n \times n$  **adjacency matrix**  $\mathbf{A}$  with elements

$$A_{ij} = \begin{cases} 1 & \text{if } (i, j) \in \mathcal{E} \\ 0 & \text{if } (i, j) \notin \mathcal{E} \end{cases}$$

where  $n = |\mathcal{V}|$

- For undirected graphs  $\mathbf{A}$  is a symmetric matrix, i.e.  $\mathbf{A} = \mathbf{A}^T$
- For weighted graphs we often store the **connectivity matrix** or **cost-adjacency matrix**,  $\mathbf{C}$ , where

$$C_{ij} = \begin{cases} w_{ij} & \text{if } (i, j) \in \mathcal{E} \\ 0 & \text{if } (i, j) \notin \mathcal{E} \end{cases}$$

# Adjacency Lists

- For **dense** graphs where the number of edges is  $\Theta(n^2)$  the adjacency matrix is often a useful representation
- But in **sparse** graphs where the number of edges is  $\Theta(n)$  the adjacency matrix has a very large number of zeros
- A more efficient representation is in terms of the adjacency list where the set of outgoing edges is stored for each node
- In some applications it is useful to store both the adjacency matrix and the adjacency list

# Adjacency Lists

- For **dense** graphs where the number of edges is  $\Theta(n^2)$  the adjacency matrix is often a useful representation
- But in **sparse** graphs where the number of edges is  $\Theta(n)$  the adjacency matrix has a very large number of zeros
- A more efficient representation is in terms of the adjacency list where the set of outgoing edges is stored for each node
- In some applications it is useful to store both the adjacency matrix and the adjacency list

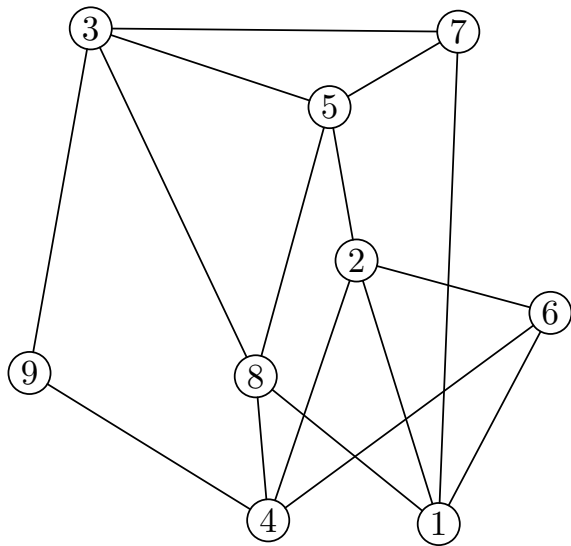
# Adjacency Lists

- For **dense** graphs where the number of edges is  $\Theta(n^2)$  the adjacency matrix is often a useful representation
- But in **sparse** graphs where the number of edges is  $\Theta(n)$  the adjacency matrix has a very large number of zeros
- A more efficient representation is in terms of the adjacency list where the set of outgoing edges is stored for each node
- In some applications it is useful to store both the adjacency matrix and the adjacency list

# Adjacency Lists

- For **dense** graphs where the number of edges is  $\Theta(n^2)$  the adjacency matrix is often a useful representation
- But in **sparse** graphs where the number of edges is  $\Theta(n)$  the adjacency matrix has a very large number of zeros
- A more efficient representation is in terms of the adjacency list where the set of outgoing edges is stored for each node
- In some applications it is useful to store both the adjacency matrix and the adjacency list

# Representing Undirected Graphs

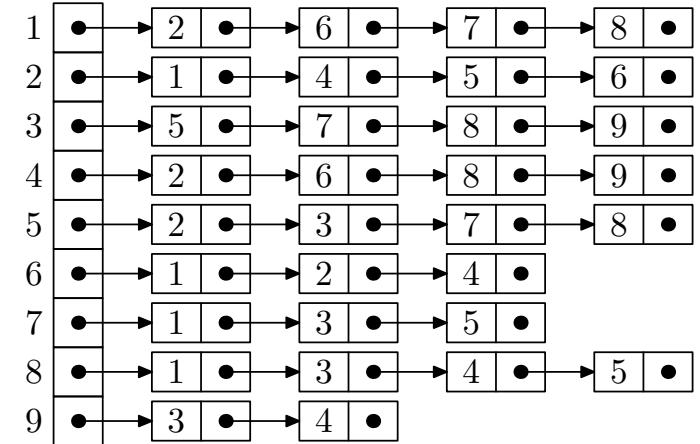


Graph

from

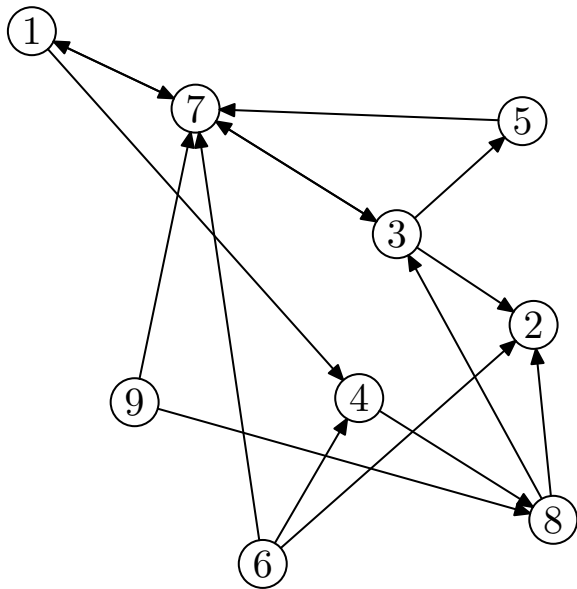
	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	1	1	1	0
2	1	0	0	1	1	1	0	0	0
3	0	0	0	0	1	0	1	1	1
4	0	1	0	0	0	1	0	1	1
5	0	1	1	0	0	0	1	1	0
6	1	1	0	1	0	0	0	0	0
7	1	0	1	0	1	0	0	0	0
8	1	0	1	1	1	0	0	0	0
9	0	0	1	1	0	0	0	0	0

Adjacency Matrix



Adjacency List

# Representing Digraphs



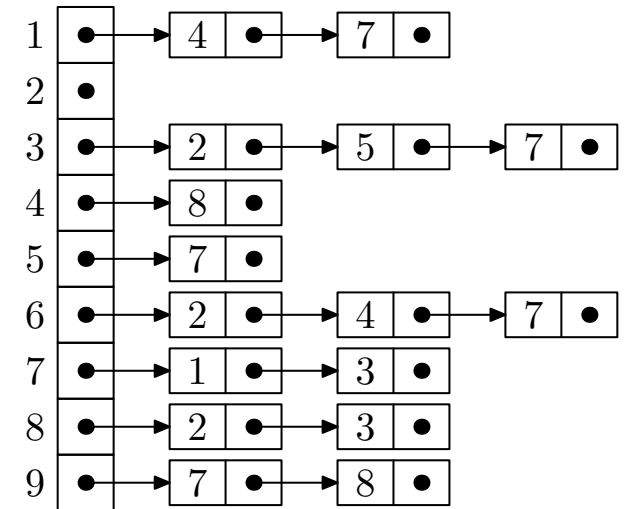
Graph

from

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	1	0	0
2	0	0	1	0	0	1	0	1	0
3	0	0	0	0	0	0	1	1	0
4	1	0	0	0	0	1	0	0	0
5	0	0	1	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	1	0	1	0	1	1	0	0	1
8	0	0	0	1	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0

to

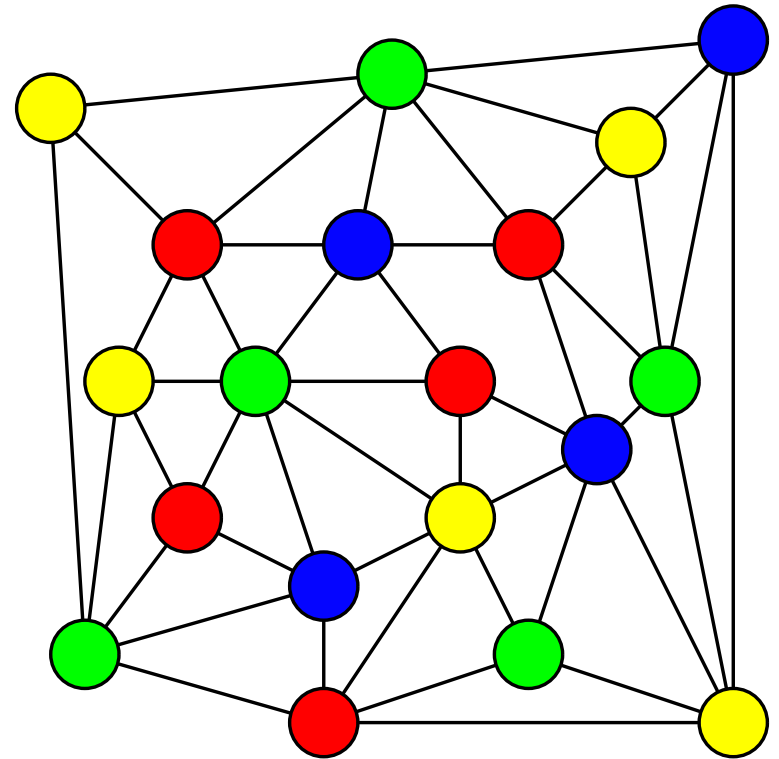
Adjacency Matrix



Adjacency List

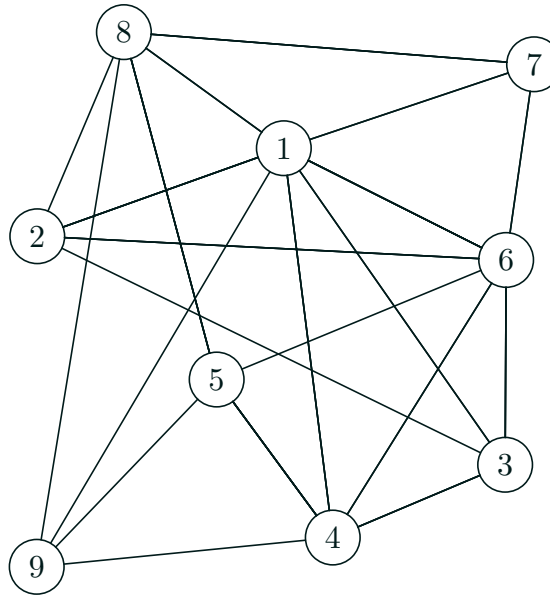
# Outline

1. Graph Theory
2. Applications of Graphs
  - Geometric applications
  - Relational applications
3. Implementing Graphs
4. **Graph Problems**



# Hamilton Cycle

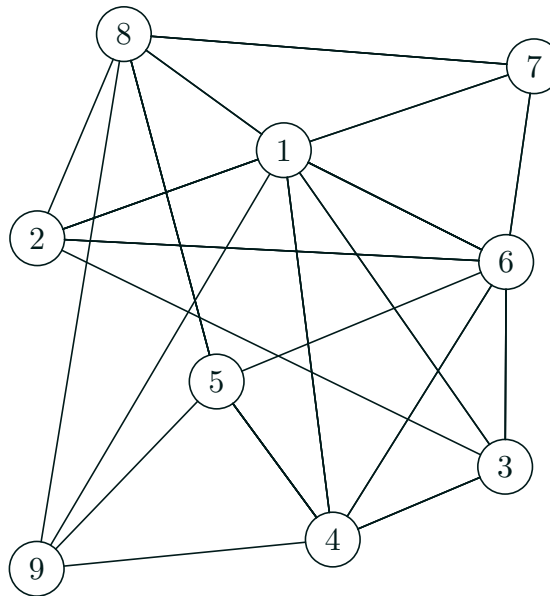
- The Euler path problem is to find a path through a multigraph that passes through every edge once—easy to solve
- The Hamilton cycle problem is to find a cycle that goes through each vertex exactly once



- There is no known efficient algorithm to solve this

# Hamilton Cycle

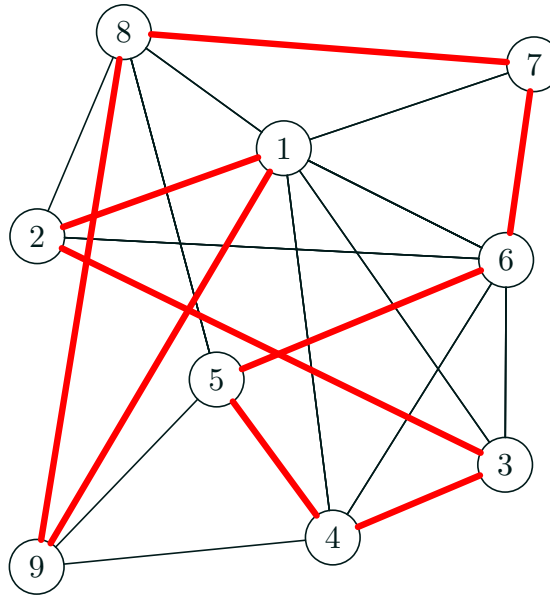
- The Euler path problem is to find a path through a multigraph that passes through every edge once—easy to solve
- The Hamilton cycle problem is to find a cycle that goes through each vertex exactly once



- There is no known efficient algorithm to solve this

# Hamilton Cycle

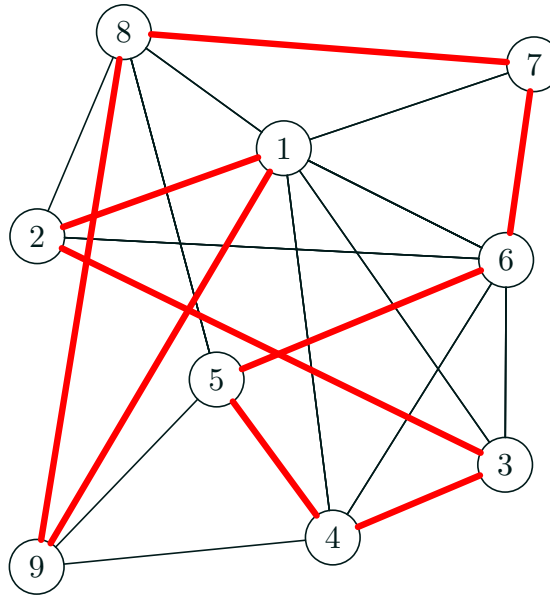
- The Euler path problem is to find a path through a multigraph that passes through every edge once—easy to solve
- The Hamilton cycle problem is to find a cycle that goes through each vertex exactly once



- There is no known efficient algorithm to solve this

# Hamilton Cycle

- The Euler path problem is to find a path through a multigraph that passes through every edge once—easy to solve
- The Hamilton cycle problem is to find a cycle that goes through each vertex exactly once



- There is no known efficient algorithm to solve this

# Shortest Path and TSP

- The shortest path problem is to find a path between two nodes
- There is an efficient algorithm—see next lecture
- In the travelling salesperson problem the task is to find the shortest tour (Hamilton cycle)—we usually assume there is an edge between every pair of nodes
- There is no known efficient algorithm to solve all TSPs

# Shortest Path and TSP

- The shortest path problem is to find a path between two nodes
- There is an efficient algorithm—see next lecture
- In the travelling salesperson problem the task is to find the shortest tour (Hamilton cycle)—we usually assume there is an edge between every pair of nodes
- There is no known efficient algorithm to solve all TSPs

# Shortest Path and TSP

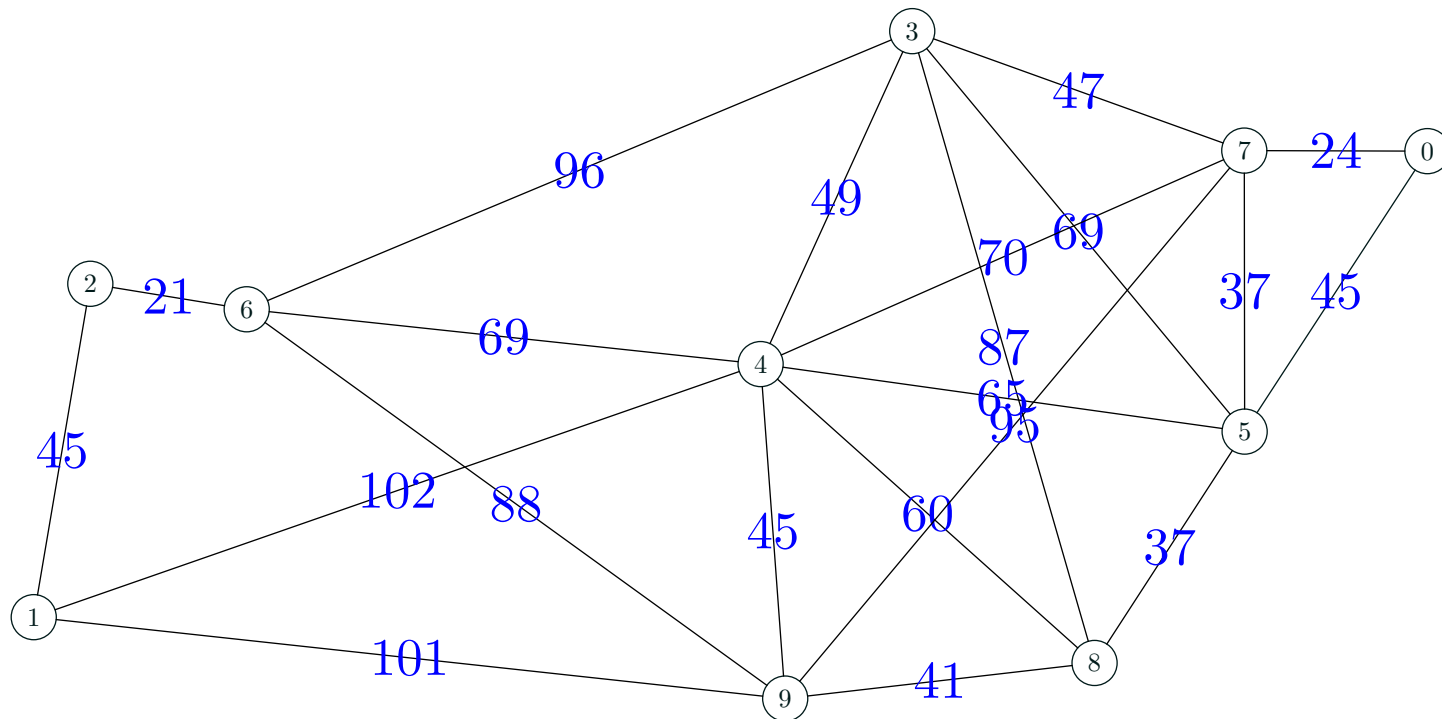
- The shortest path problem is to find a path between two nodes
- There is an efficient algorithm—see next lecture
- In the travelling salesperson problem the task is to find the shortest tour (Hamilton cycle)—we usually assume there is an edge between every pair of nodes
- There is no known efficient algorithm to solve all TSPs

# Shortest Path and TSP

- The shortest path problem is to find a path between two nodes
- There is an efficient algorithm—see next lecture
- In the travelling salesperson problem the task is to find the shortest tour (Hamilton cycle)—we usually assume there is an edge between every pair of nodes
- There is no known efficient algorithm to solve all TSPs

# Minimum Spanning Tree

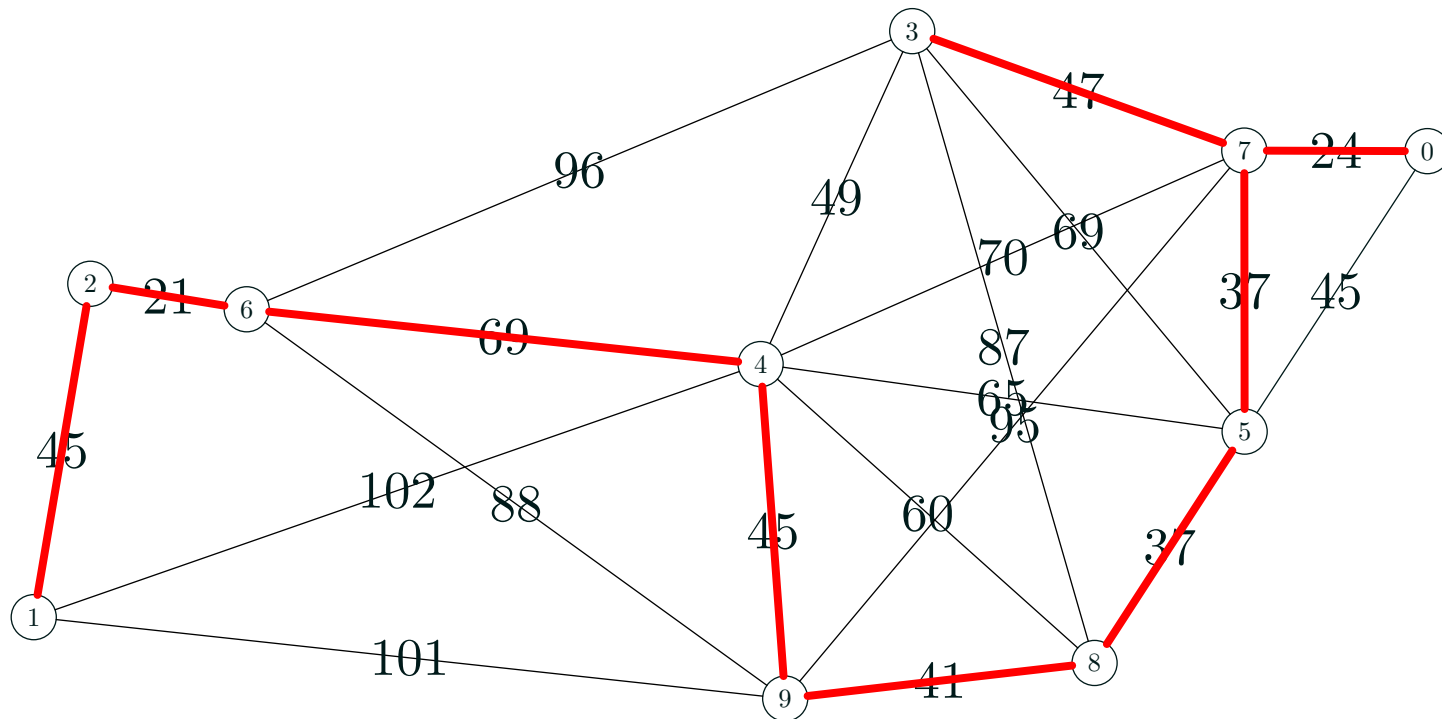
- Suppose we want to construct pylons connecting a number of cities using the least amount of cable



- We will study an efficient algorithm to solve this in the next but one lecture

# Minimum Spanning Tree

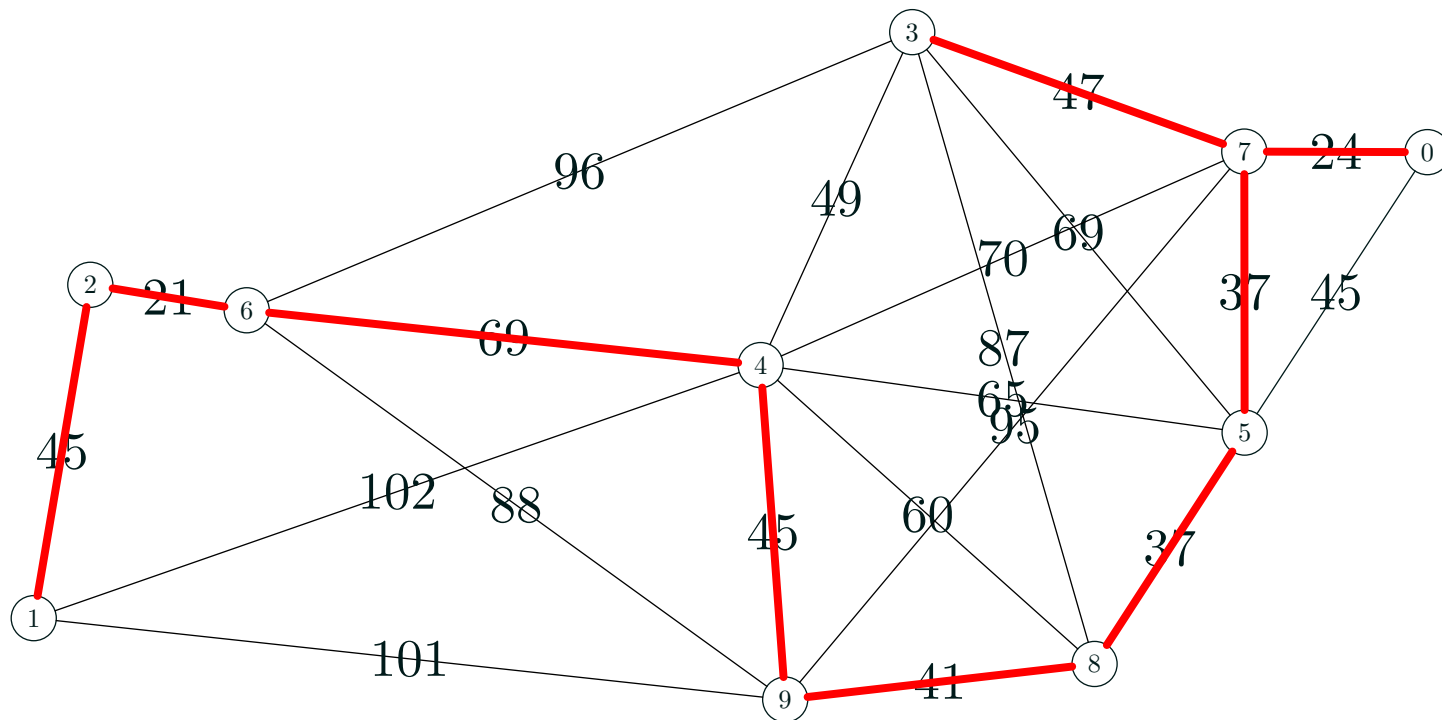
- Suppose we want to construct pylons connecting a number of cities using the least amount of cable



- We will study an efficient algorithm to solve this in the next but one lecture

# Minimum Spanning Tree

- Suppose we want to construct pylons connecting a number of cities using the least amount of cable



- We will study an efficient algorithm to solve this in the next but one lecture

# Graph Partitioning

- The simplest version of this problem is to cut a graph into two equal halves so that you minimise the number of edges you cut
- If the edges are weighted then you want to minimise the sum of edges that are cut
- If the vertices are weighted you want to balance the sum of vertex weights in the two partitions
- An example of this problem is in dividing up a problem to run on a parallel computer
  - ★ Nodes are subtasks (weights on nodes are run times)
  - ★ Edge weights indicate communication cost
- There is no known efficient algorithm to solve this

# Graph Partitioning

- The simplest version of this problem is to cut a graph into two equal halves so that you minimise the number of edges you cut
- If the edges are weighted then you want to minimise the sum of edges that are cut
- If the vertices are weighted you want to balance the sum of vertex weights in the two partitions
- An example of this problem is in dividing up a problem to run on a parallel computer
  - ★ Nodes are subtasks (weights on nodes are run times)
  - ★ Edge weights indicate communication cost
- There is no known efficient algorithm to solve this

# Graph Partitioning

- The simplest version of this problem is to cut a graph into two equal halves so that you minimise the number of edges you cut
- If the edges are weighted then you want to minimise the sum of edges that are cut
- If the vertices are weighted you want to balance the sum of vertex weights in the two partitions
- An example of this problem is in dividing up a problem to run on a parallel computer
  - ★ Nodes are subtasks (weights on nodes are run times)
  - ★ Edge weights indicate communication cost
- There is no known efficient algorithm to solve this

# Graph Partitioning

- The simplest version of this problem is to cut a graph into two equal halves so that you minimise the number of edges you cut
- If the edges are weighted then you want to minimise the sum of edges that are cut
- If the vertices are weighted you want to balance the sum of vertex weights in the two partitions
- An example of this problem is in dividing up a problem to run on a parallel computer
  - ★ Nodes are subtasks (weights on nodes are run times)
  - ★ Edge weights indicate communication cost
- There is no known efficient algorithm to solve this

# Graph Partitioning

- The simplest version of this problem is to cut a graph into two equal halves so that you minimise the number of edges you cut
- If the edges are weighted then you want to minimise the sum of edges that are cut
- If the vertices are weighted you want to balance the sum of vertex weights in the two partitions
- An example of this problem is in dividing up a problem to run on a parallel computer
  - ★ Nodes are subtasks (weights on nodes are run times)
  - ★ Edge weights indicate communication cost
- There is no known efficient algorithm to solve this

# Graph Partitioning

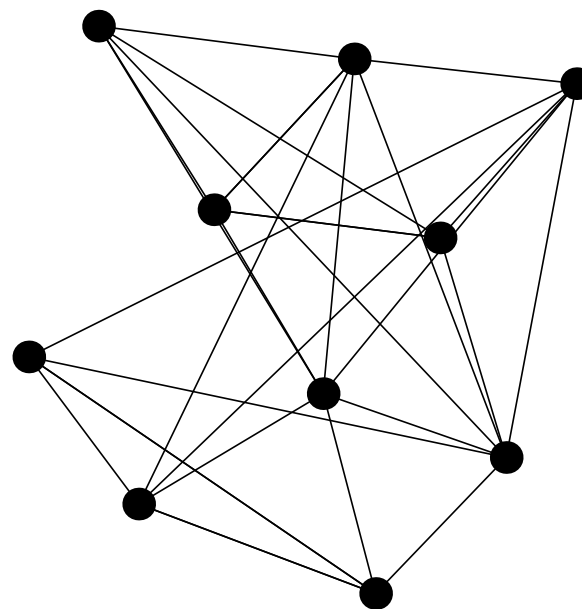
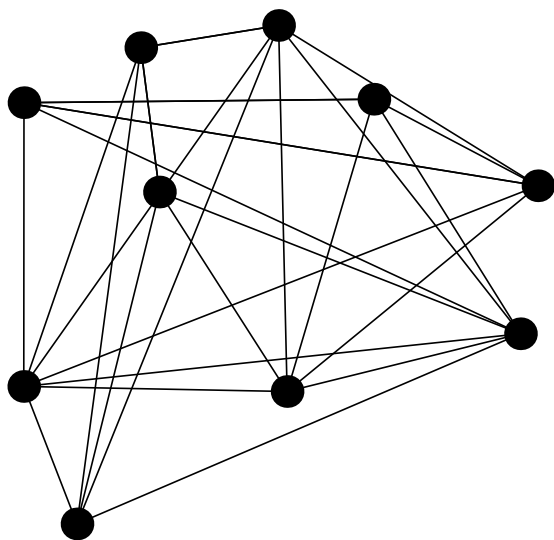
- The simplest version of this problem is to cut a graph into two equal halves so that you minimise the number of edges you cut
- If the edges are weighted then you want to minimise the sum of edges that are cut
- If the vertices are weighted you want to balance the sum of vertex weights in the two partitions
- An example of this problem is in dividing up a problem to run on a parallel computer
  - ★ Nodes are subtasks (weights on nodes are run times)
  - ★ Edge weights indicate communication cost
- There is no known efficient algorithm to solve this

# Graph Partitioning

- The simplest version of this problem is to cut a graph into two equal halves so that you minimise the number of edges you cut
- If the edges are weighted then you want to minimise the sum of edges that are cut
- If the vertices are weighted you want to balance the sum of vertex weights in the two partitions
- An example of this problem is in dividing up a problem to run on a parallel computer
  - ★ Nodes are subtasks (weights on nodes are run times)
  - ★ Edge weights indicate communication cost
- There is no known efficient algorithm to solve this

# Graph Isomorphism

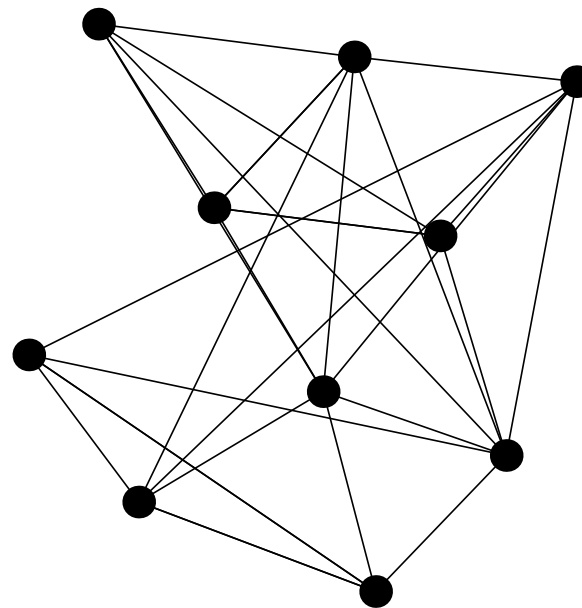
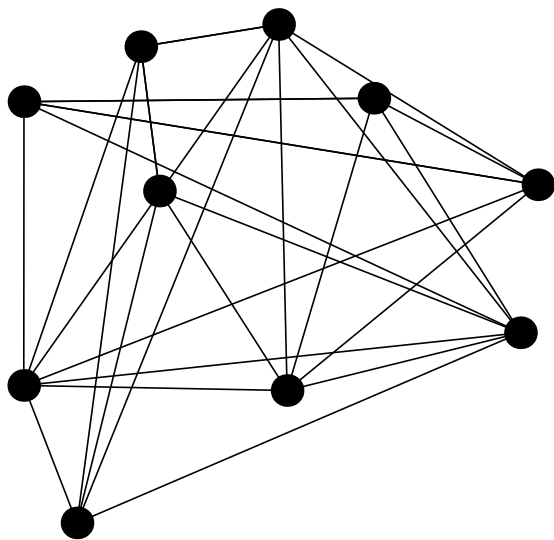
- Do two graphs have the same structure?



- There is no known efficient algorithm to solve this problem
- Theoretically it is interesting because it is not NP-complete

# Graph Isomorphism

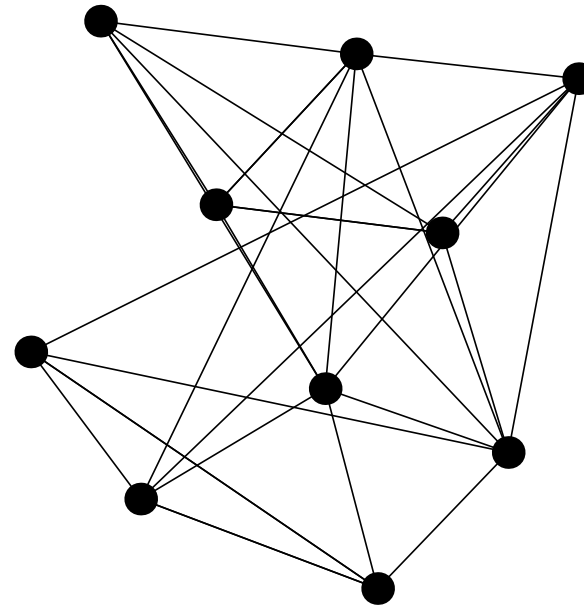
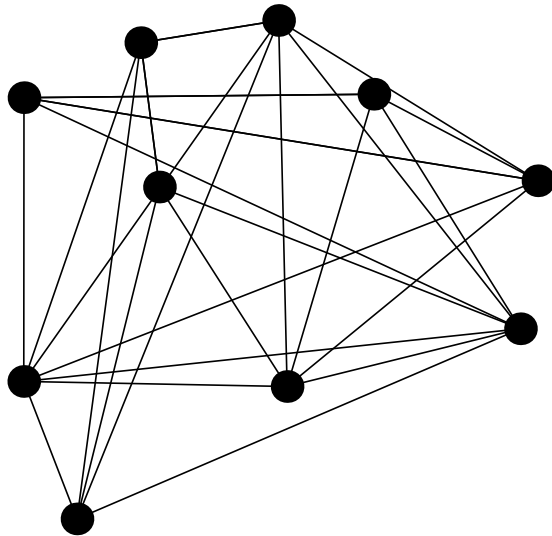
- Do two graphs have the same structure?



- There is no known efficient algorithm to solve this problem
- Theoretically it is interesting because it is not NP-complete

# Graph Isomorphism

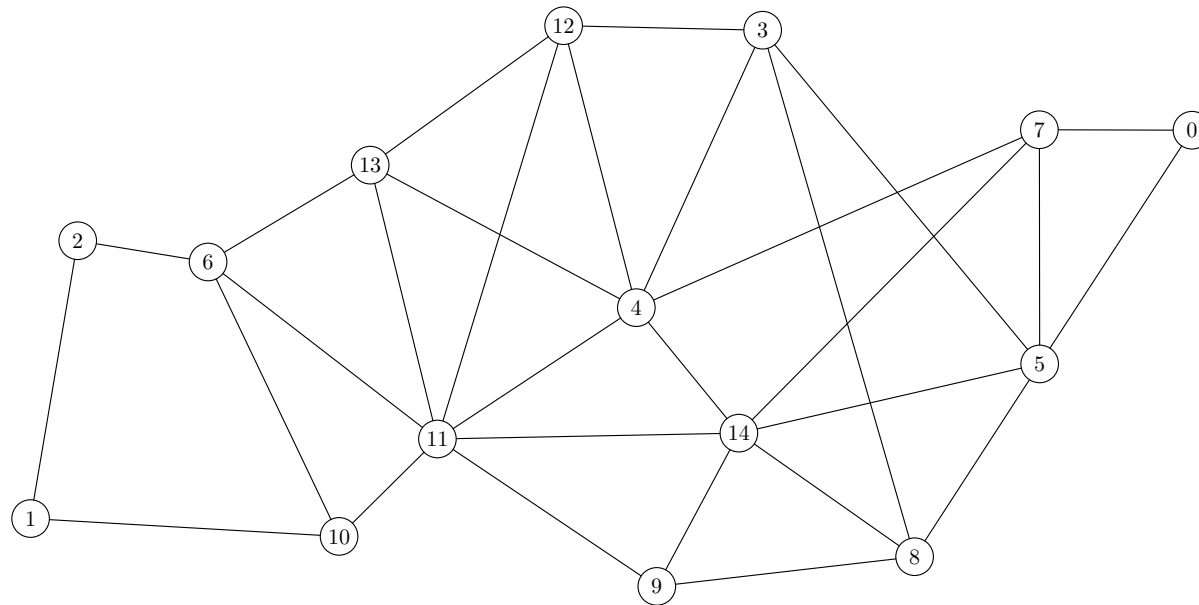
- Do two graphs have the same structure?



- There is no known efficient algorithm to solve this problem
- Theoretically it is interesting because it is not NP-complete

# Vertex Cover

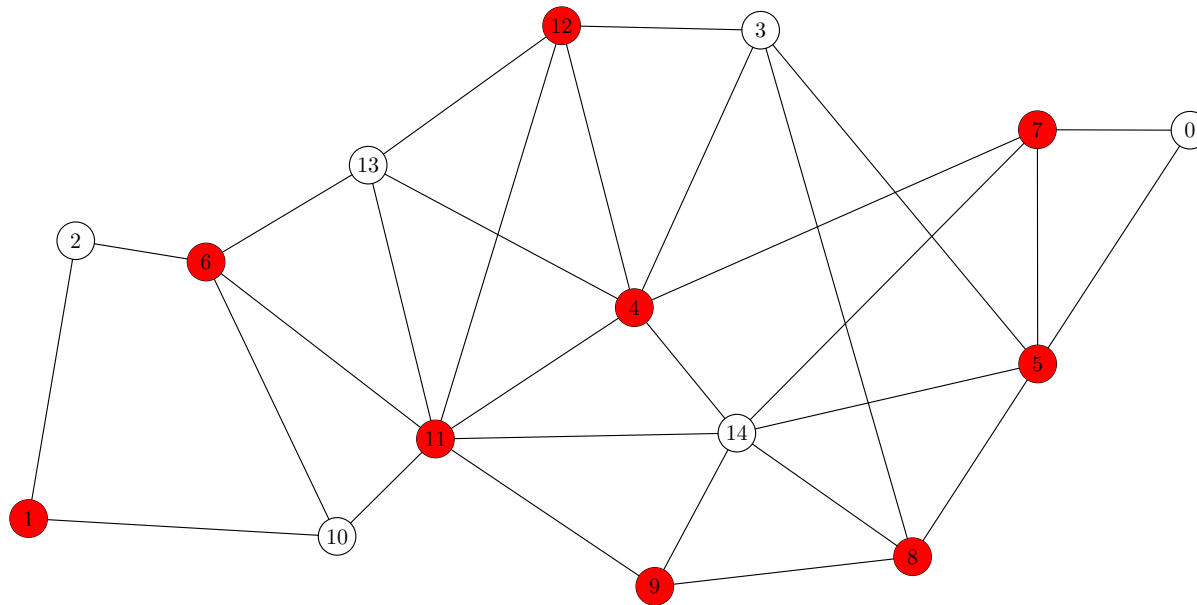
- How many guards do you need to cover all the corridors in a museum



- There is no known efficient algorithm to solve this

# Vertex Cover

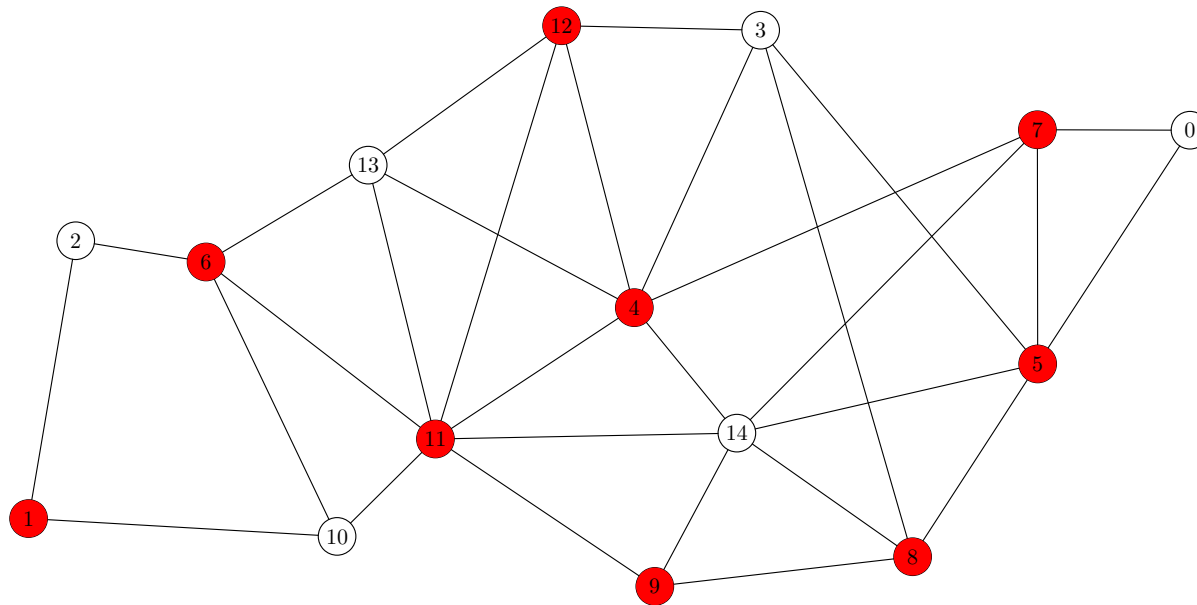
- How many guards do you need to cover all the corridors in a museum



- There is no known efficient algorithm to solve this

# Vertex Cover

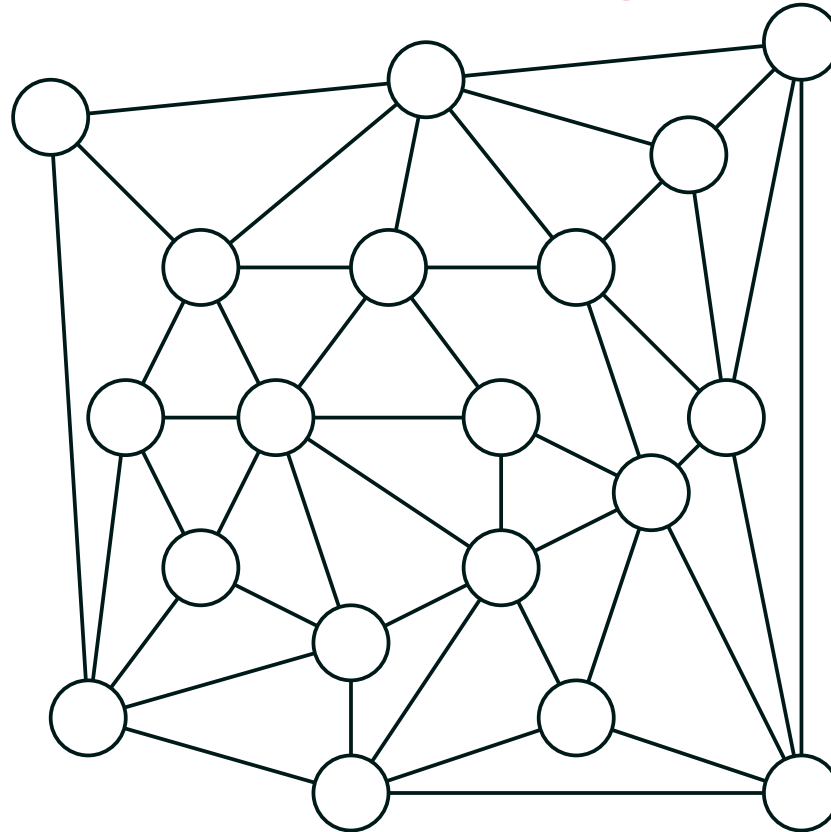
- How many guards do you need to cover all the corridors in a museum



- There is no known efficient algorithm to solve this

# Graph Colouring

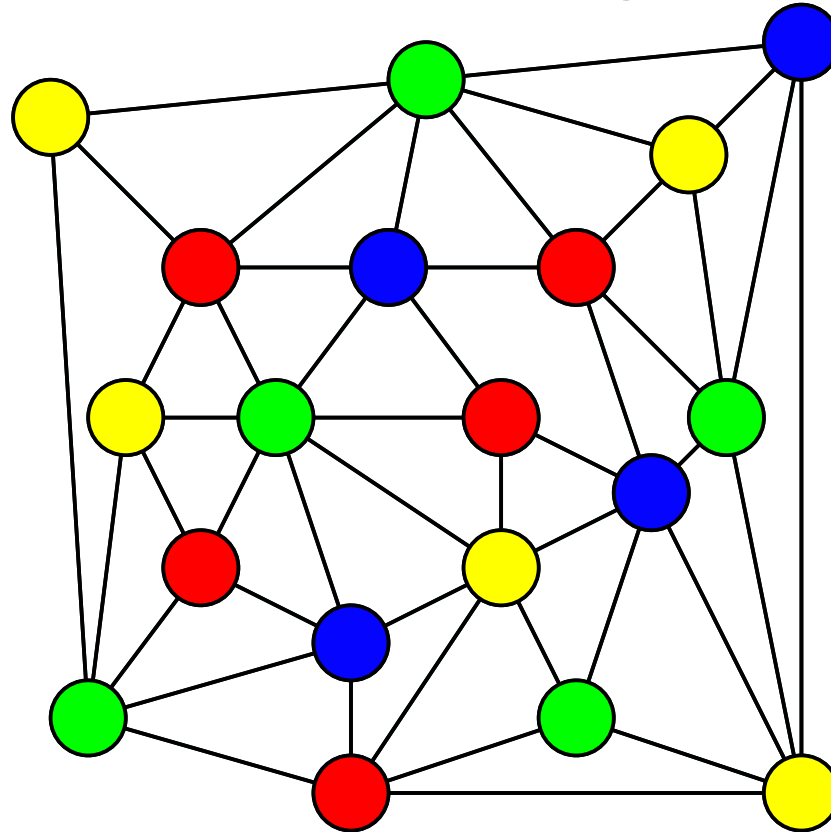
- How many colours do I need to colour a graph with no conflicts



- There is no known efficient algorithm to solve this

# Graph Colouring

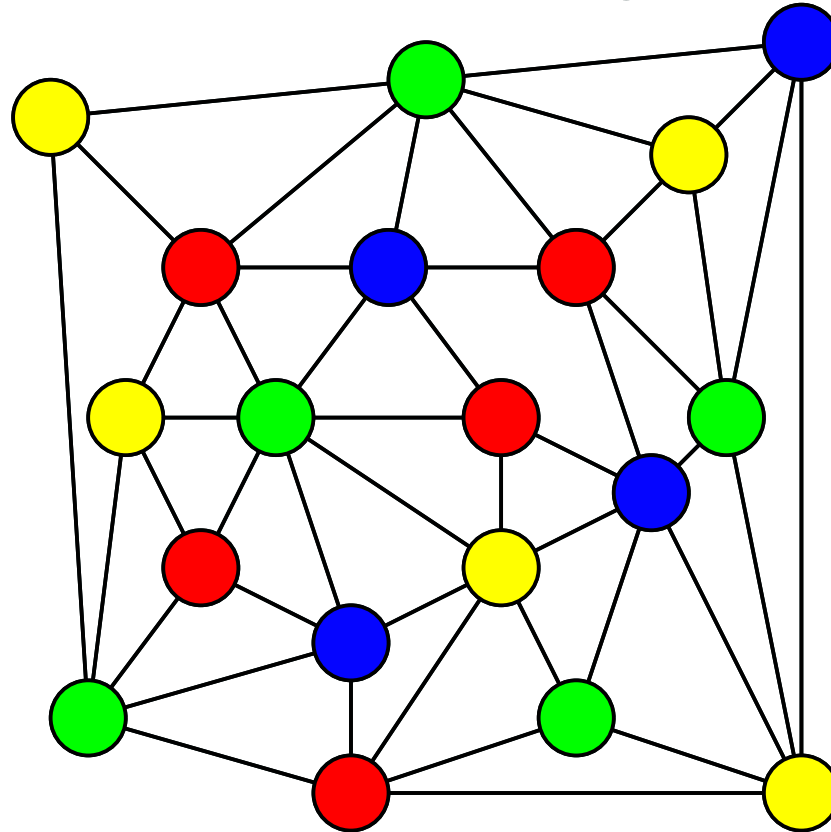
- How many colours do I need to colour a graph with no conflicts



- There is no known efficient algorithm to solve this

# Graph Colouring

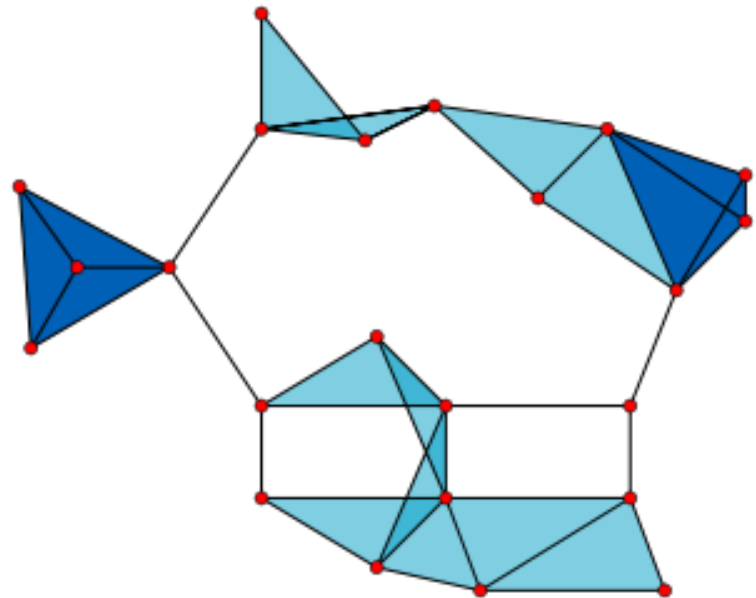
- How many colours do I need to colour a graph with no conflicts



- There is no known efficient algorithm to solve this

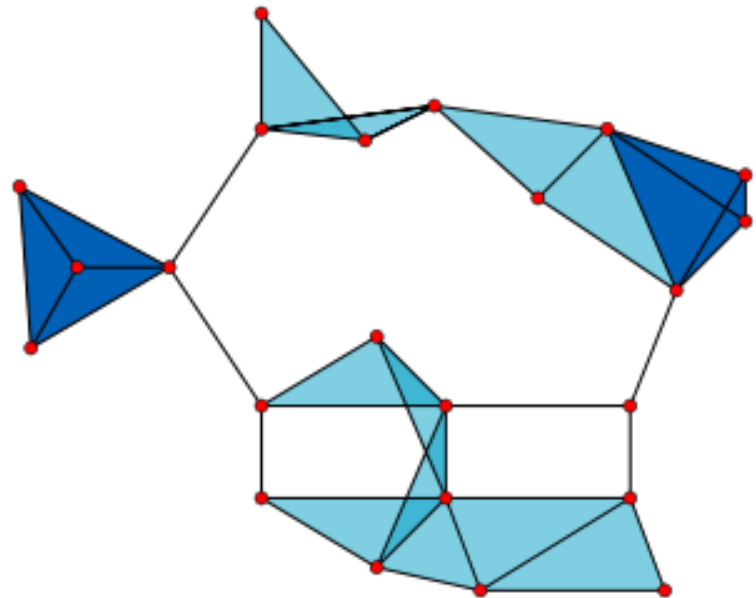
# Other Graph Problems

- These are only a sample of the many famous graph problems
- Others include
  - ★ Max-clique (hard)
  - ★ Maximal independent set (hard)
  - ★ Maximal flow problem (easy)
  - ★ Max-cut (hard)



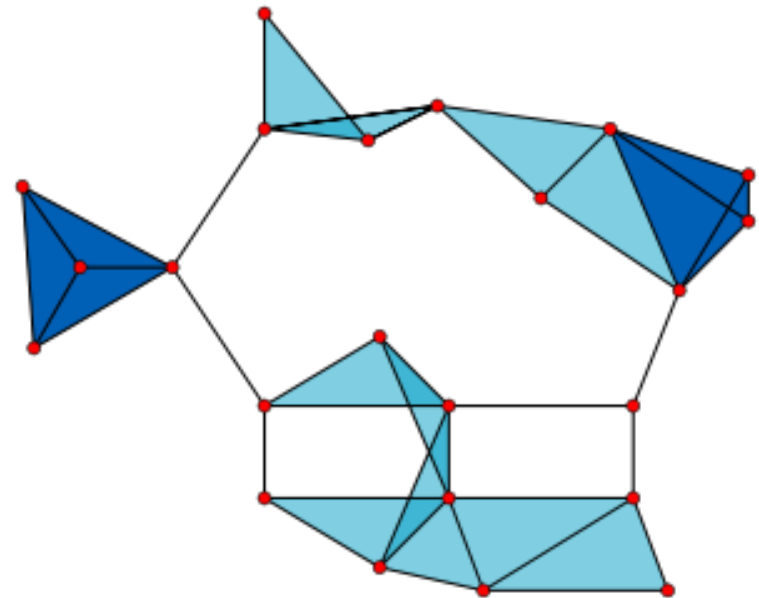
# Other Graph Problems

- These are only a sample of the many famous graph problems
- Others include
  - ★ Max-clique (hard)
  - ★ Maximal independent set (hard)
  - ★ Maximal flow problem (easy)
  - ★ Max-cut (hard)



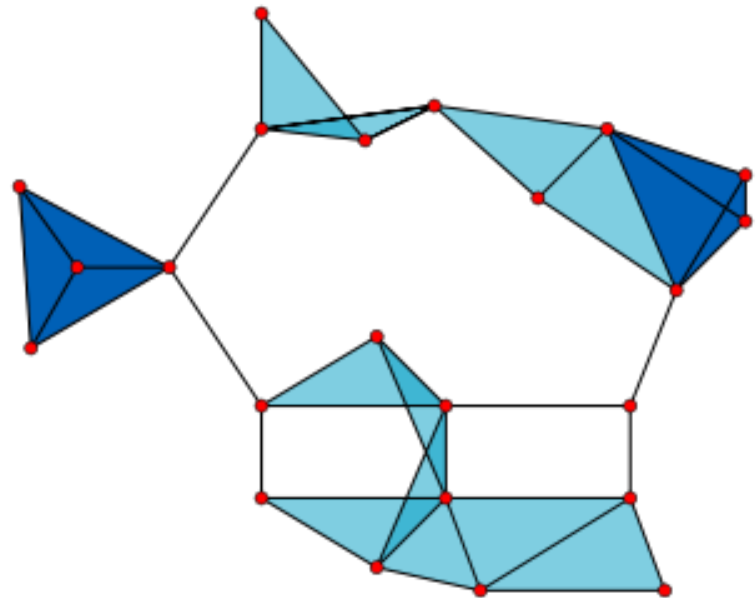
# Other Graph Problems

- These are only a sample of the many famous graph problems
- Others include
  - ★ Max-clique (hard)
  - ★ Maximal independent set (hard)
  - ★ Maximal flow problem (easy)
  - ★ Max-cut (hard)



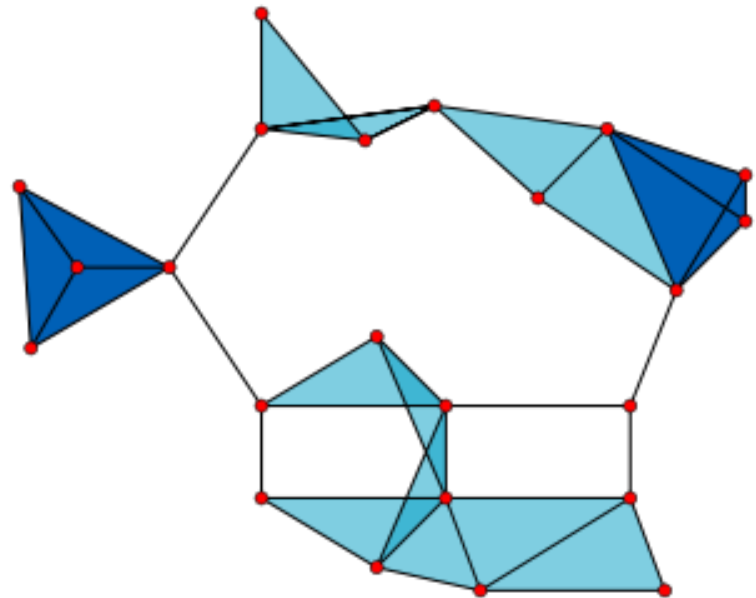
# Other Graph Problems

- These are only a sample of the many famous graph problems
- Others include
  - ★ Max-clique (hard)
  - ★ Maximal independent set (hard)
  - ★ Maximal flow problem (easy)
  - ★ Max-cut (hard)



# Other Graph Problems

- These are only a sample of the many famous graph problems
- Others include
  - ★ Max-clique (hard)
  - ★ Maximal independent set (hard)
  - ★ Maximal flow problem (easy)
  - ★ Max-cut (hard)



# Lessons

- Graphs are an important method for abstracting problems
- They appear in a huge number of disparate fields
- There are many problems for which efficient algorithms are known
- There are many problems which are believed to be hard—i.e. there aren't any efficient algorithms
- Even for hard problems there are good algorithms for finding approximated solutions

# Lessons

- Graphs are an important method for abstracting problems
- They appear in a huge number of disparate fields
- There are many problems for which efficient algorithms are known
- There are many problems which are believed to be hard—i.e. there aren't any efficient algorithms
- Even for hard problems there are good algorithms for finding approximated solutions

# Lessons

- Graphs are an important method for abstracting problems
- They appear in a huge number of disparate fields
- There are many problems for which efficient algorithms are known
- There are many problems which are believed to be hard—i.e. there aren't any efficient algorithms
- Even for hard problems there are good algorithms for finding approximated solutions

# Lessons

- Graphs are an important method for abstracting problems
- They appear in a huge number of disparate fields
- There are many problems for which efficient algorithms are known
- There are many problems which are believed to be hard—i.e. there aren't any efficient algorithms
- Even for hard problems there are good algorithms for finding approximated solutions

# Lessons

- Graphs are an important method for abstracting problems
- They appear in a huge number of disparate fields
- There are many problems for which efficient algorithms are known
- There are many problems which are believed to be hard—i.e. there aren't any efficient algorithms
- Even for hard problems there are good algorithms for finding approximated solutions