Chapter 2: Graph Theory

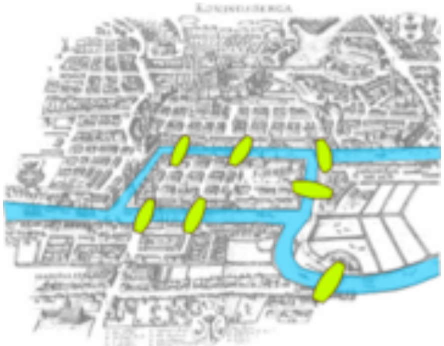
Graph Theory Introduction

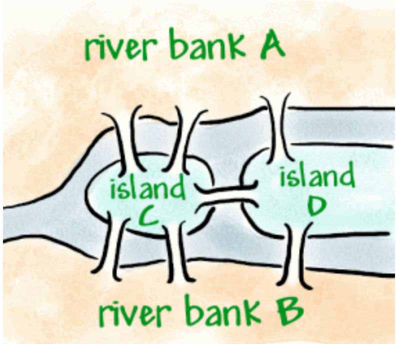
Applications of Graphs:

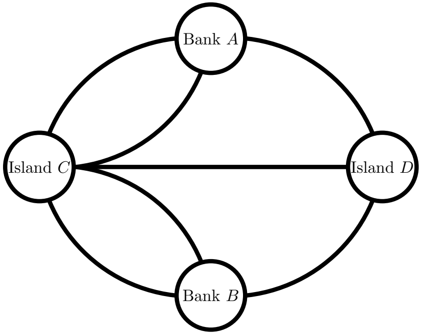
- Convenient representation/visualisation to many Mathematical, Engineering and Science Problems.
- Fundamental Data Structure in Computer Science
- Many examples to follow

Graph Theory History: The Königsberg bridge problem

- Solved by Euler (1707-1783).
- Popular Problem of its day
- Map of Königsberg







- The Königsberg bridge problem was the following:

  Is it possible to cross each of the seven bridges of Königsberg exactly once and return to the starting point?

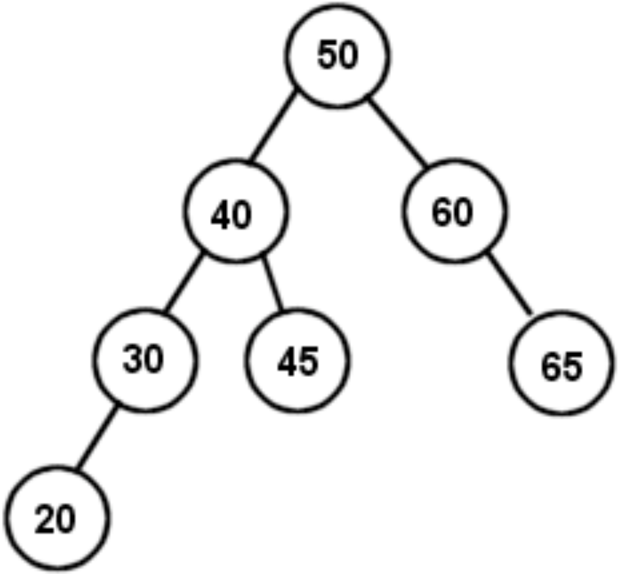
- We return to this later.

Some Computer Science

- Graphs and Trees: **Fundamental Data Structures**

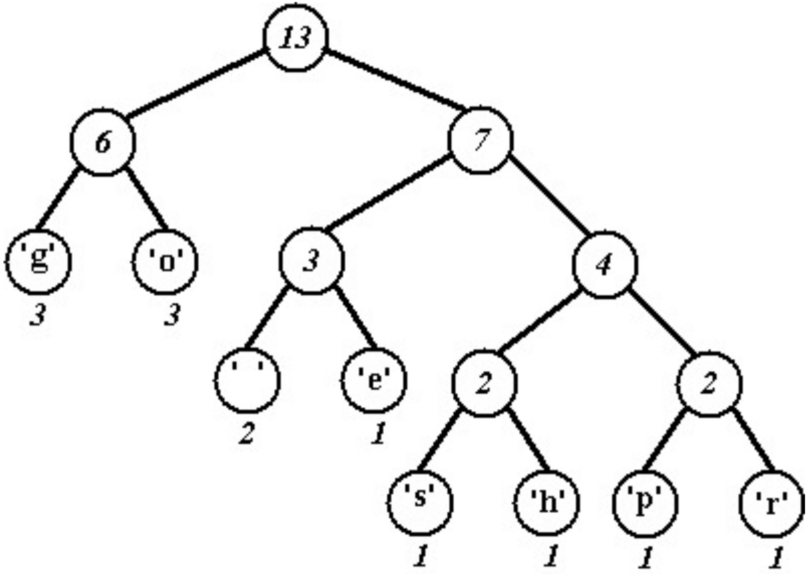
  Used in all branches or Computer Science
- Sorting and Searching Algorithms
- Knowledge Representation: Database, Data Mining
- Computer Networks: Internet, Mobile Comms, Networking
- Data Compression/Coding
- Artificial Intelligence
  - Knowledge Representation and Reasoning, Game Playing, Planning, Natural Language
- Computer Graphics/Image Processing/Computer Vision
- Compilers and *Many Many More* . . . . . .

Graphs and Networks Example: Sorting



Binary Tree Sort — very common data structure/used in many algorithms

Graphs and Networks Example: Compression/Coding



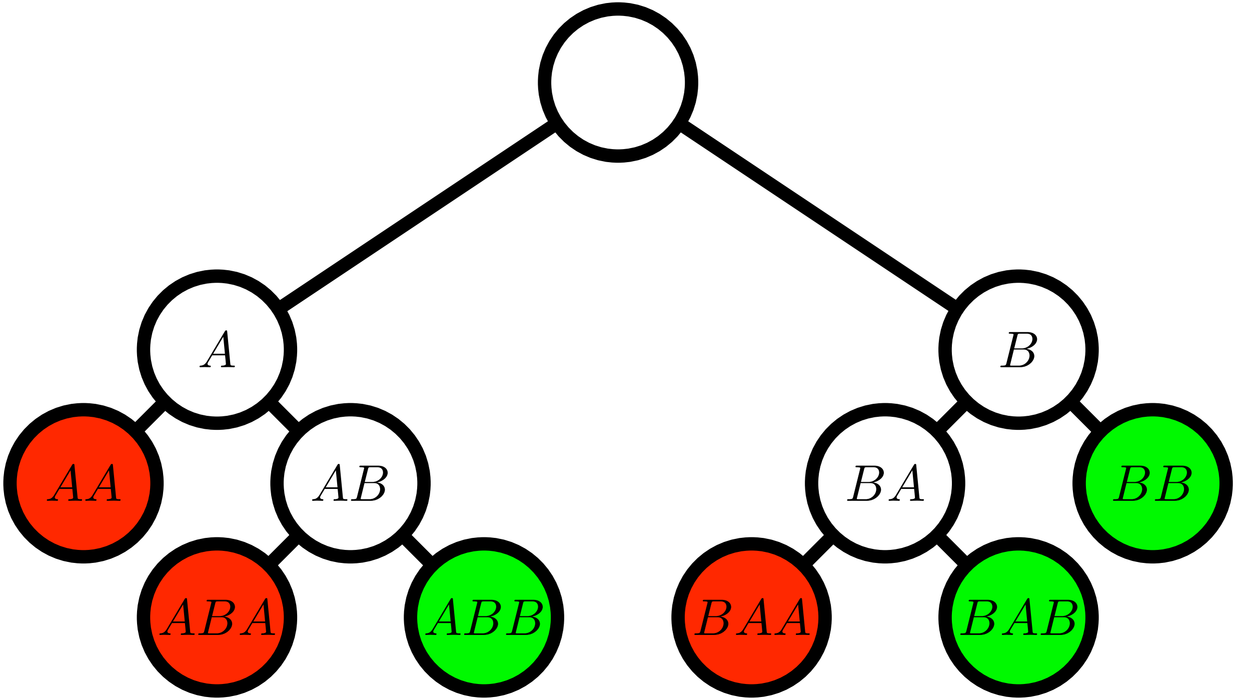
Codes:

char	binary
'g'	00
'o'	01
'p'	1110
'h'	1101
'e'	101
'r'	1111
's'	1100
' '	100

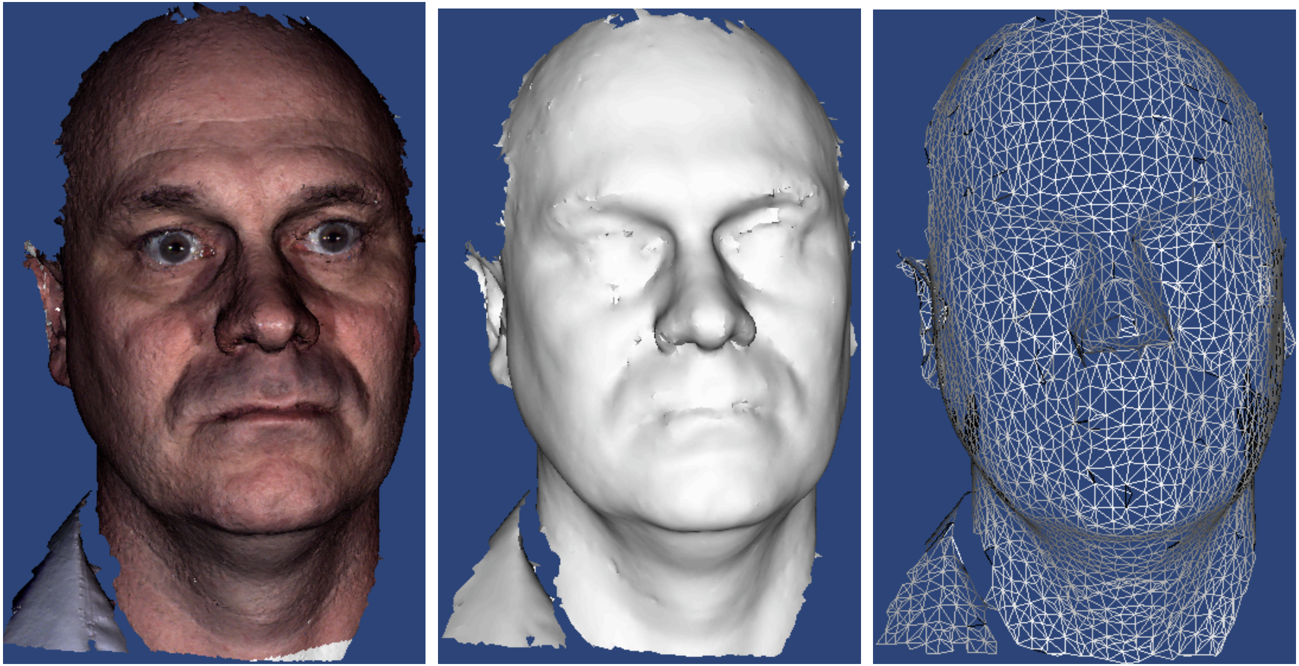
- Count number of occurrences of tokens (characters here) in a sequence.
- Sort then in a tree then, Code via tree traversal
- We return to this later.

Graphs and Networks Example: Game Playing

Best of Three Sets Tennis Match Representation:



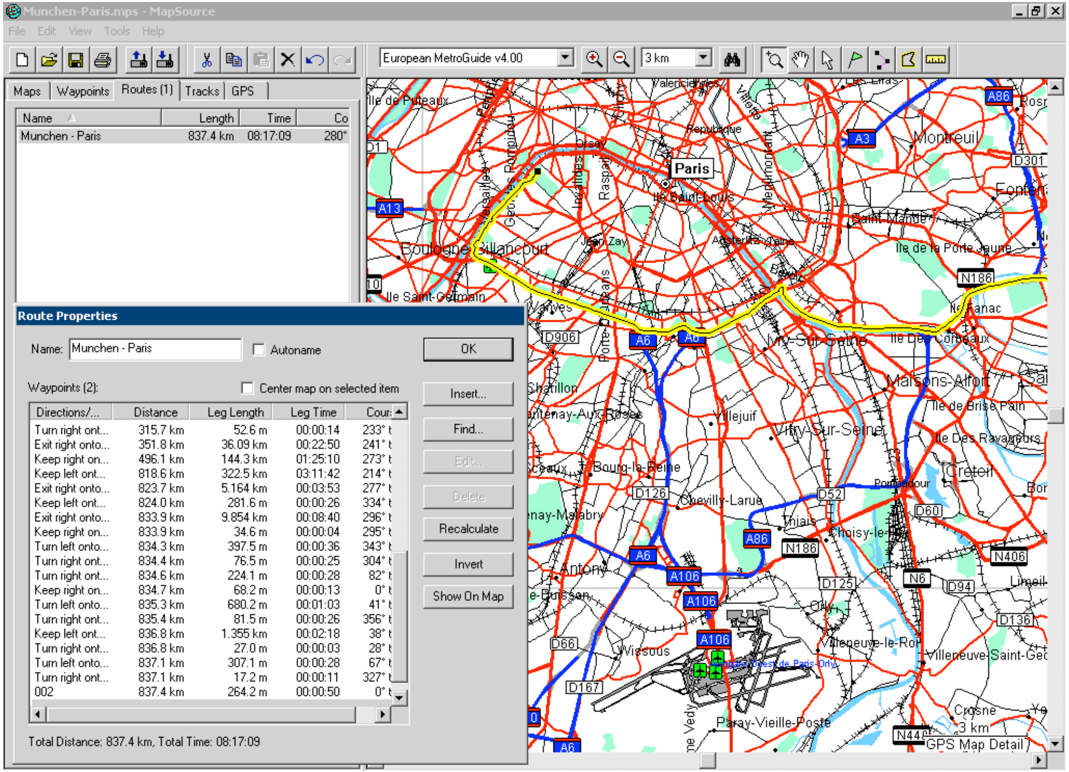
Graphs and Networks Example: Computer Graphics



Fundamental 3D computer graphics structure — 3D Mesh

Connectivity and Adjacency essential for topology and geometric structure.

Graphs and Networks Example: Route Planning



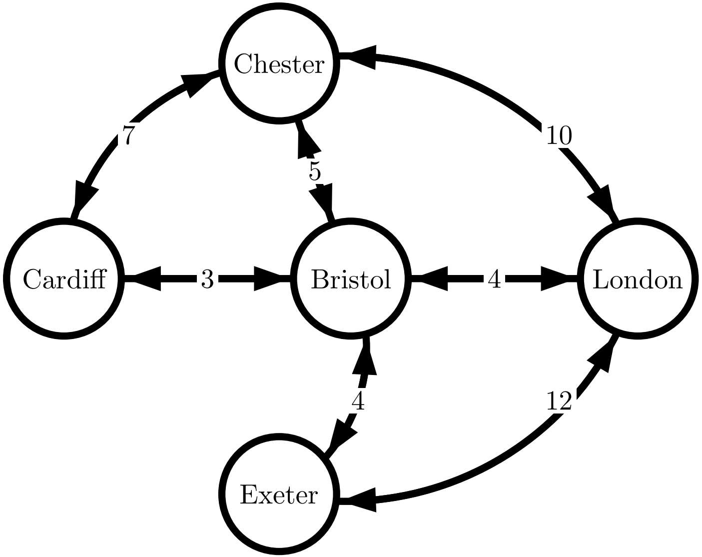
or **AA Route Planner** or similar.

CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

CM0167 Maths For Comp. Sci.

Graphs and Networks Example: Route Planning

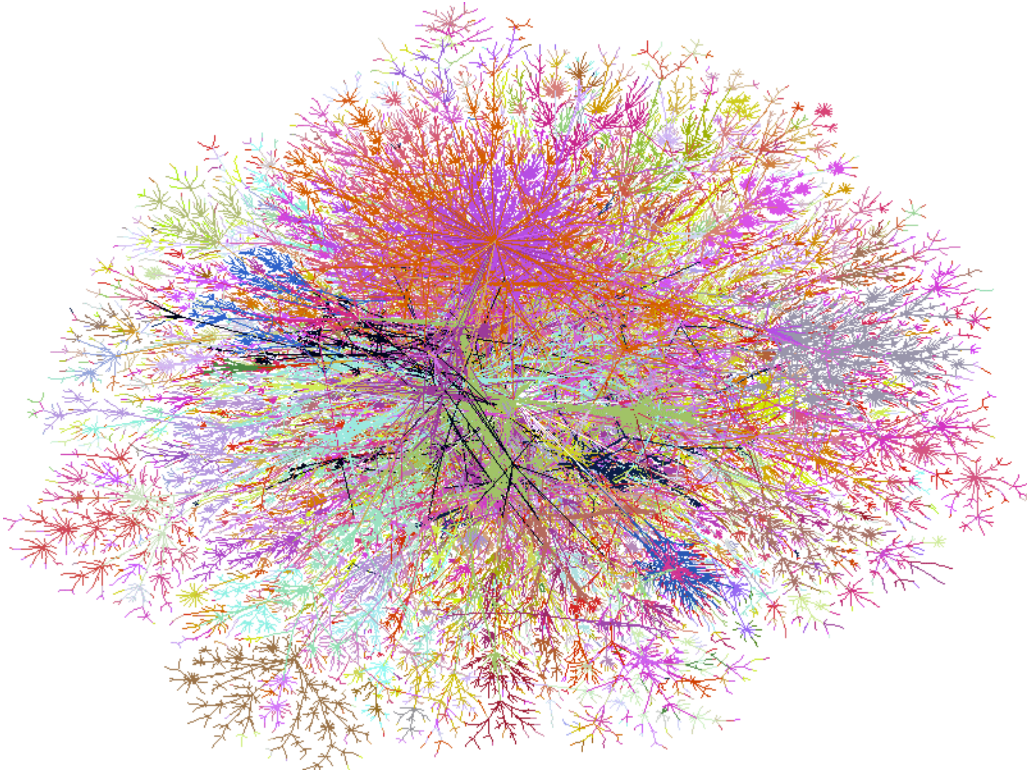
Classic Example of Shortest Path of a Graph



- Each path has a cost (distance/average time) to destination
- Find shortest path = fastest route.
- We return to this later.

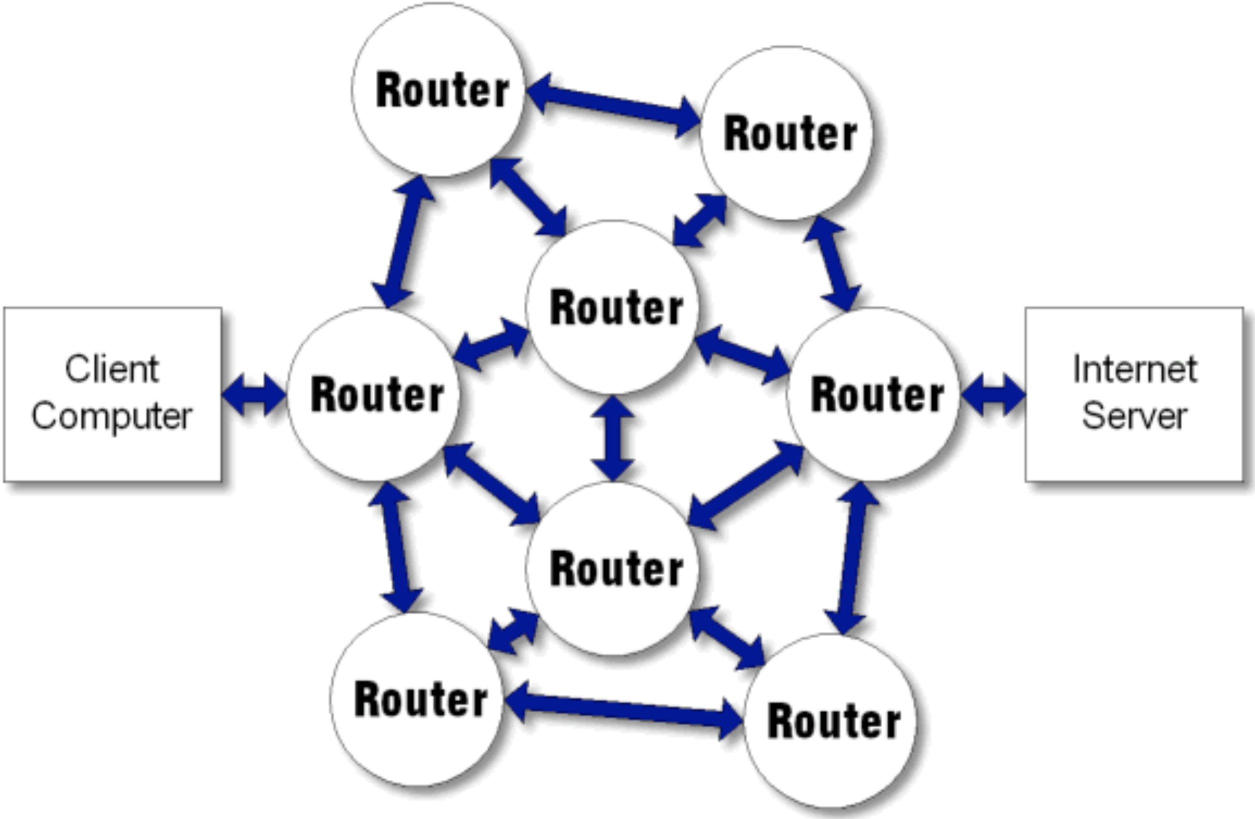
Graphs and Networks Example: Internet Network

The Internet as a Large Graph:



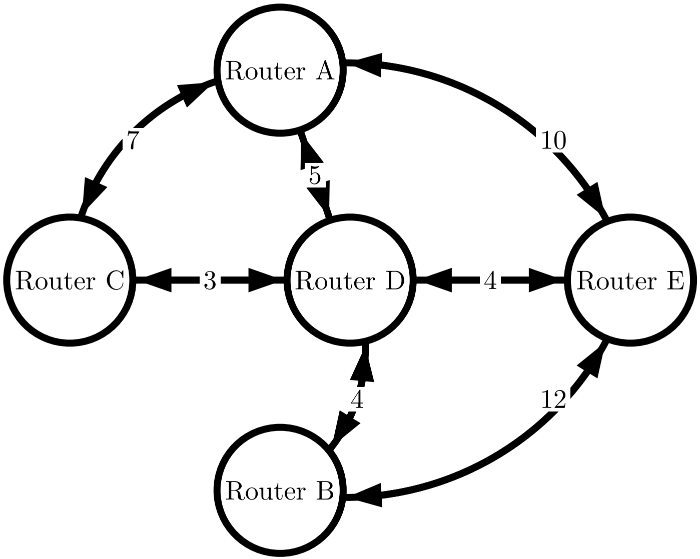
Internet Mapping Project

Graphs and Networks Example: Internet Routing



Graphs and Networks Example: Internet Planning

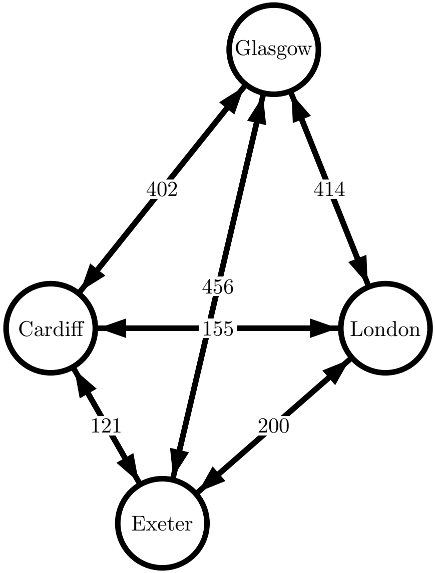
Classic Example of Shortest Path of a Graph



- Same problem as in route finding
- Find shortest path = best/fastest route on internet
- We return to this later.

Graphs and Networks Example: Travelling Salesperson Problem (TSP)

Classic Optimisation Problem

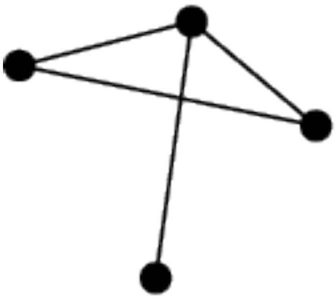


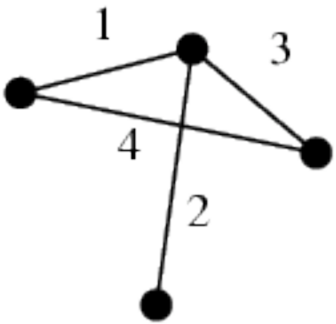
- Similar problem as in route
- Person must visit a number of cities in the minimum distance
- We return to this later.

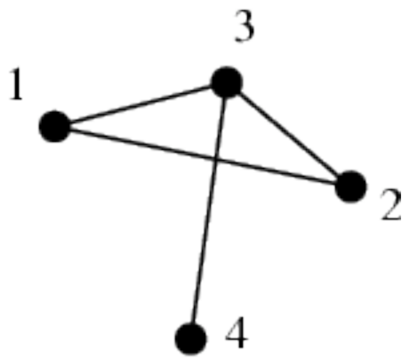
Graph Theory Basics

Definition 2.1 (Graph)**.**
A graph G consists of a set of elements called vertices and a set of elements called edges. Each edge joins two vertices.

Graphs are usually labelled:
Vertices and/or Edges can be labelled.







Why Label Graphs?

Vertices:

- To give semantic meaning e.g. Places to visit in TSP or Autoroute
- Labels can be arbitrary or change to prove some relationship between graphs **more soon**
- When we describe edges we usually refer to sets of vertices **more soon**

Why Label Graphs?

Edges:

- We use graphs to represent data, encode knowledge or enforce relationships between data
- Numbers usually represent weights, distances or cost of some relationship between the 2 vertices
- Graph Theory enumerates these weights in many ways to attempt to solve a problem:
  - Minimum cost — shortest path **more soon**
  - Maximum cost **more soon**
  - Max-Min costs in game playing **more soon**

Definition 2.2 (Weighted Graph, Weighted Digraph).

A weighted graph G is a graph where each edge connecting two vertices is assigned a weight. This weight is often interpreted as a distance or some other cost of travelling between the vertices.

A weighted digraph D is a digraph where each arc is assigned a weight. This weight is often interpreted as a distance or some other cost of travelling between the vertices.

We will see some examples of labels and weights very soon. But first we need yet more definitions!

Mathematical Notation

We denote a graph, G as set of Vertices, V, and Edges, E, as follows:

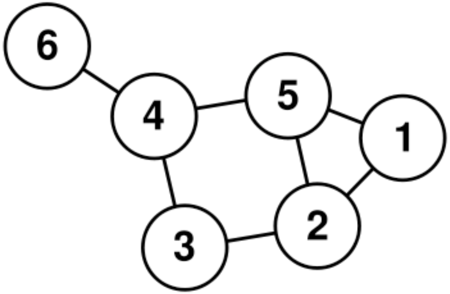
$$G = (V, E)$$

We may also write the vertex set for a graph, G, as $V(G)$

Similarly the Edge set for a graph, G, as $E(G)$

We often describe the Edges as collection or set of labelled Vertices that describe the endpoints or connections in the graph:

Example 2.1 (Vertex and Edge Sets).



A labelled simple graph with vertex set: $V = \{1, 2, 3, 4, 5, 6\}$ *and edge set:* $E = \{\{1, 2\}, \{1, 5\}, \{2, 3\}, \{2, 5\}, \{3, 4\}, \{4, 5\}, \{4, 6\}\}$

E may also be written as $E = \{e_1, e_2, e_3, \ldots\}$ where e_1, e_2 etc. are endpoint sets, e.g. $e_1 = \{1, 2\}, e_2 = \{1, 5\}, \ldots$

E is also sometimes written as $E = \{12, 15, 23, 25, 34, 45, 46\}$

Back
Close

Order of a Graph: Vertex and Edge Cardinality

Definition 2.3 (Order of a Graph).
The cardinality of V, is also called the order of graph, G, is defined to be:

The number of vertices in V.

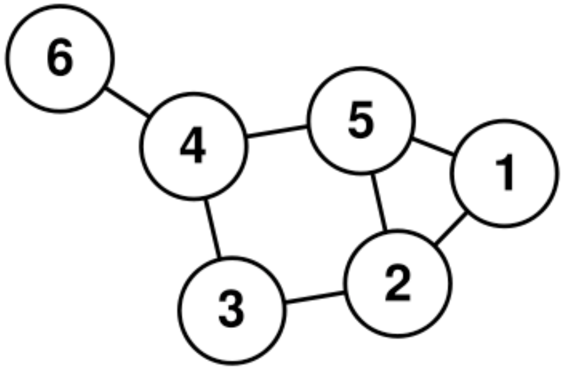
- This is denoted by $|V|$.
- We usually use n to denote the order of G. *i.e.* $n = |V|$

Definition 2.4 (Size of a Graph).
The cardinality of E, is also called the size of graph, G is defined to be:

The number of edges,

- This denoted by $|E|$.
- We usually use m to denote the size of G. *i.e.* $m = |E|$

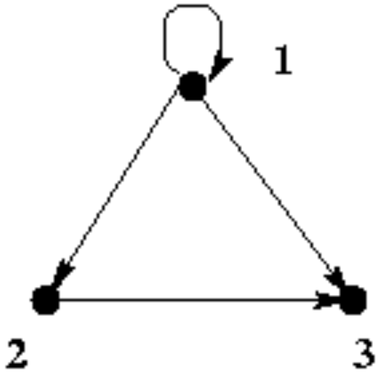
Problem 2.1 (Order and Size or a Graph).

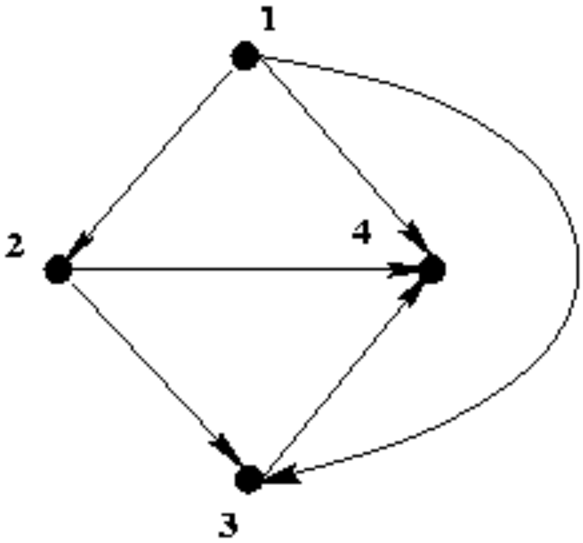


1. What is the *order* of this Graph?
2. What is the *size* of this Graph?

Definition 2.5 (Digraph)**.**

A digraph D consists of a set of elements called vertices and a set of elements called arcs. Each arc joins two vertices in a specified direction.





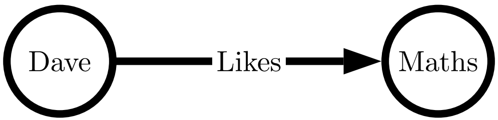
Why do we need directions in a graph

- Some relationships may be only one way.
- Relationships may differ in forward and backward direction (*Multiple Edges*)
- Directions may refer back to same end point (*Loop*)

Example 2.2 (Digraph: One Way Relationship).

Draw a graph that represents Dave like Maths

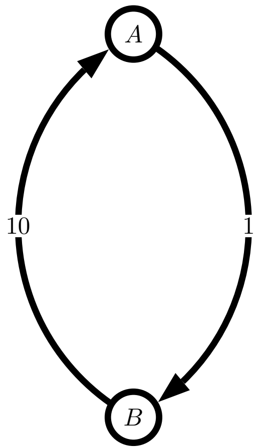
- Clearly the act of liking is a one way relationship
- Maths can't like any person but some people, e.g. Dave, can like maths.



Example 2.3 (Digraph: Two Way Relationship).

Draw a graph that represents the ease of riding a bike between two points A and B, where A is at the top of the hill and B is at the bottom of the hill

- Clearly it is easier to go from $A \to B$ than $B \to A$.
- Represent this as **weights** in two (multiple) arcs in the graph.
- Lets say it is ten times harder to ride up the hill



Example 2.4 (Digraph: Loops — Finite State Automata).

Finite State Machines/Finite State Automata

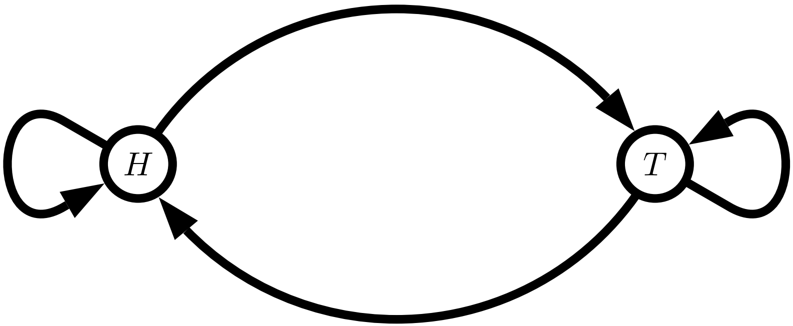
Finite State Automata area model of behavior composed of a finite number of states, transitions between those states, and actions.

Very common in many areas of Computer Science

- Speech Recognition
- Natural Language Understanding
- Theory of Computing: Formal Methods, Computability, Efficiency, Complexity
- Digital Circuits: Programmable logic device, Logic arrays
- Maths, Engineering, Biology . . .

Simple Example: Modelling a Coin Toss

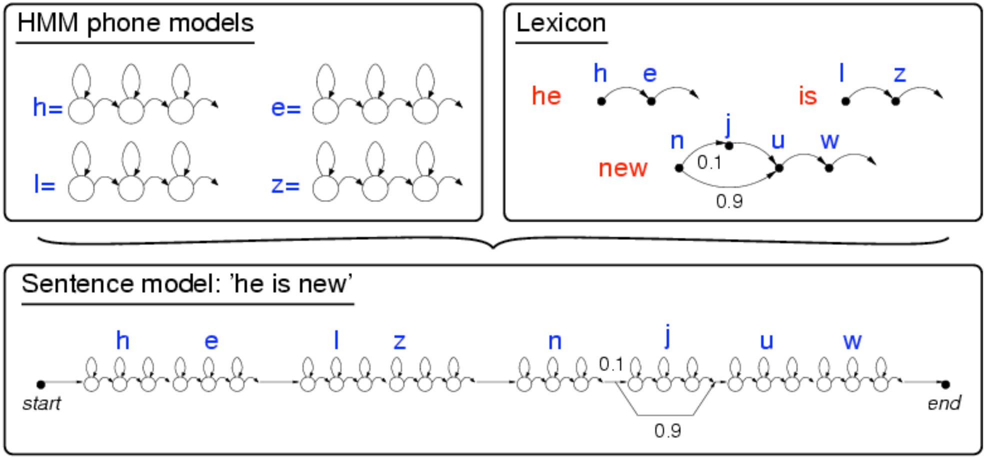
- There are Two States Only Ever: Heads (H) or Tails (T)
- Each coin toss is a finite state or event.
- Coin can either stay in same state (say another Head) or change (to Tail)



Example 2.5 (Digraph Loops: Speech Understanding).
Real World Example: Hidden Markov Models
Example of Stochastic Finite State Automata

Sample Speech features and attempt to model the pattern of speech over successive samples based on known (learned) models

- Level One — Group Speech Features into Phonemes (Phones)
- Level Two — Group Phonemes into Words
- Level Two — Group Words into Sentences



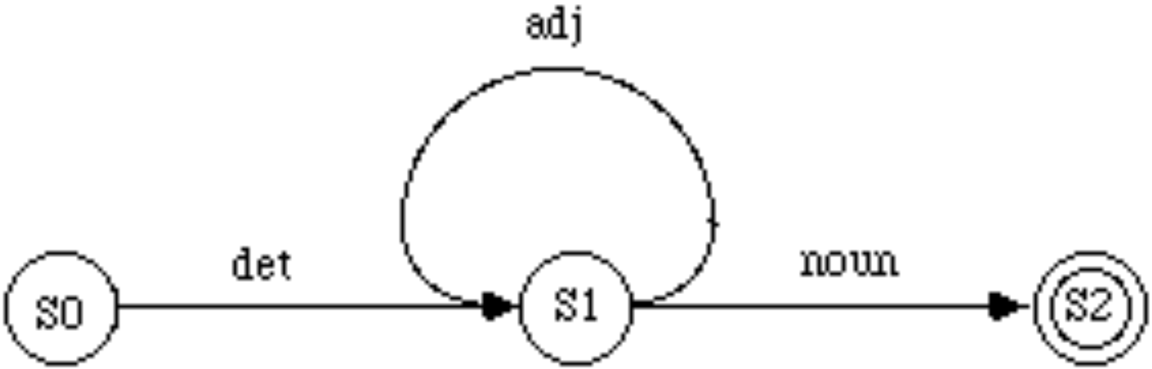
Back
Close

Example 2.6 (Digraph Loops: Linguistics).

Real World Example: Natural Language Understanding:
A large branch of Artificial Intelligence.

Representing the structure of language as a computational model.

- Can model a sentence (S) as succession of a Noun Phrase (NP) and a Verb Phase (VP): $S \to NP + VP$
- Can model a NP as digraph.
- VP similar: $VP \to V + NP$ where V is a verb
- Can decompose sentences.



Natural Language Understanding: Noun Phrase Explained

Three Parts:

Determiners : articles (the, a), demonstratives (this, that), numerals (two, five, etc.), possessives (my, their, etc.), and quantifiers (some, many, etc.); in English, determiners are usually placed before the noun;

Adjectives : (Zero?) One or more (the large cat);

Noun

Additional **Compliments** can be added to qualify noun phrases with

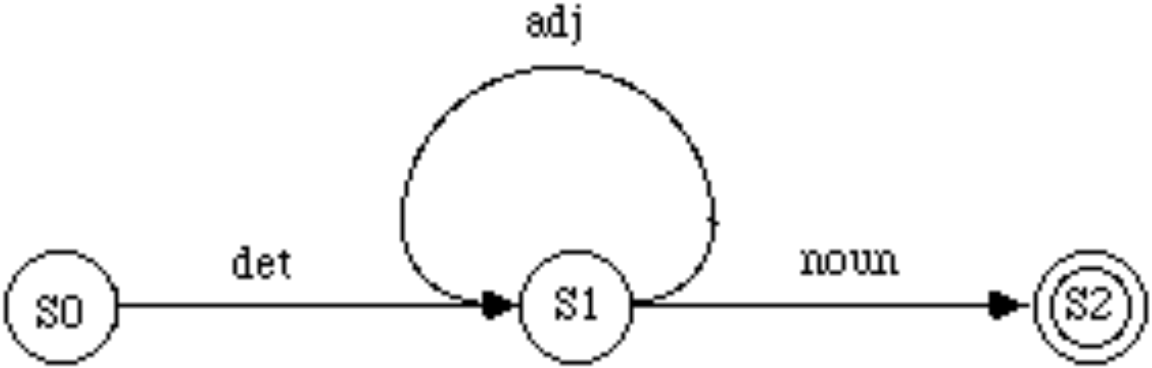
- *adpositional phrases*, such as
  `the cat with the fluffy tail`, or
- *relative clauses*, such as
  `the cat that I fed yesterday`.

but this complicates the digraph.

Back
Close

Natural Language Understanding: Noun Phrase Explained

Our Noun Phrase:



This can represent the following type of phrases:

```
the cat
the large cat
the very large cat
the very very large cat
the very very very large cat
```

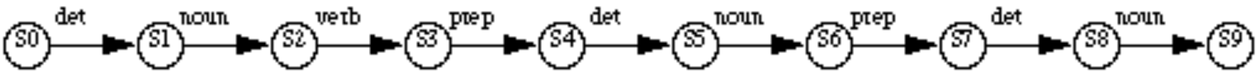
etc.

Natural Language Understanding: Decompose a sentence

Consider the sentence:

```
the cat sat on the mat by the fire
```

This might be drawn as:



This is a long *one way relationship digraph*.

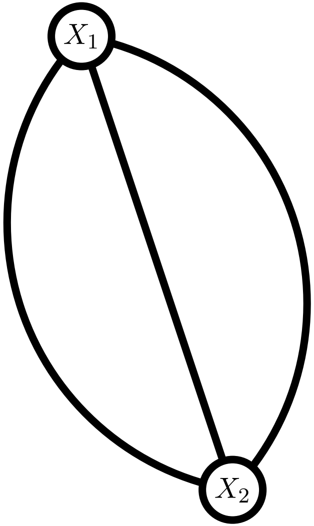
Problem 2.2 (More Advanced Natural Language Representation)**.**

- *How can the Noun Phrase digraph cope with no adjectives?*
- *Give an alternative Noun Phrase digraph representation to cope with no adjectives.*
- *Represent a Verb Phrase as a digraph.*
- *Ammend the Noun Phrase to model additional compliments.*

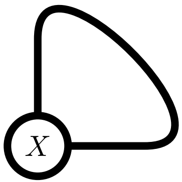
Some More Definitions

Definition 2.6 (Multiple Edges, Loops).
In a graph, two or more edges joining the same pair of vertices are multiple edges.

An edge joining a vertex to itself is a loop.



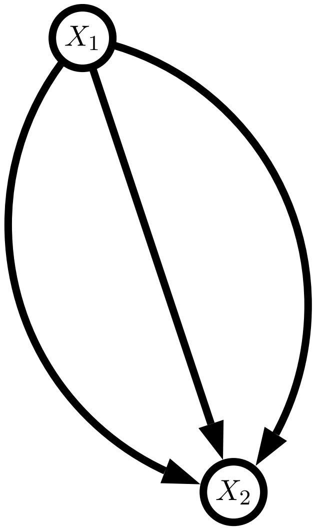
Mutliple Edges



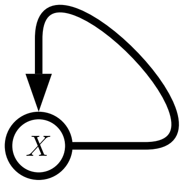
Simple Loop

Definition 2.7 (Multiple Arcs, Loops).
In a digraph, two or more arcs joining the same pair of vertices in the same direction are multiple arcs.

An arc joining a vertex to itself is a loop.



Mutliple Arcs

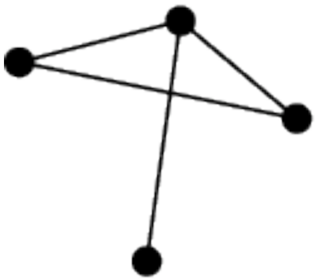


Simple Arc Loop

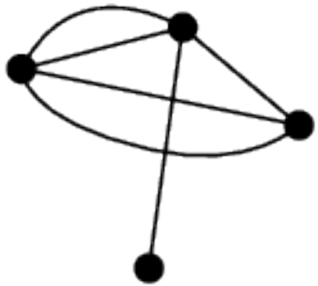
Definition 2.8 (Simple graphs, Simple digraphs).

A graph with no multiple edges or loops is a simple graph.

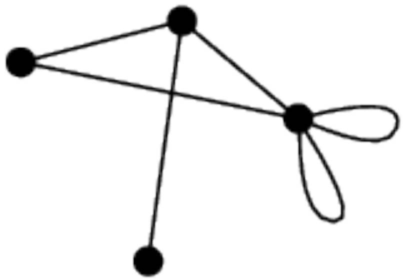
A digraph with no multiple arcs or loops is a simple digraph.



simple graph



nonsimple graph with multiple edges

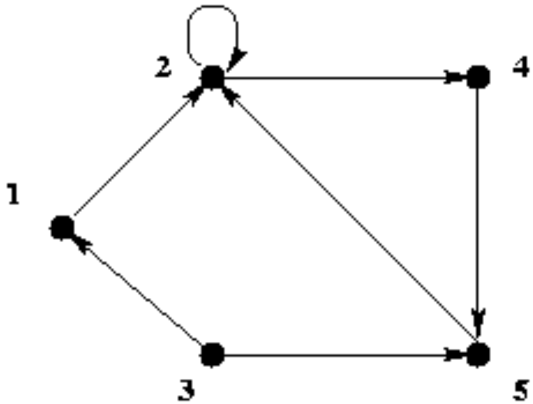


nonsimple graph with loops

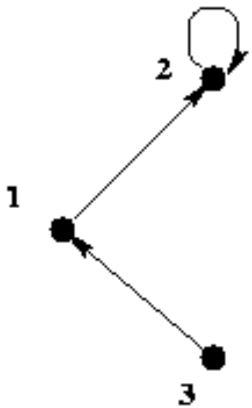
Definition 2.9 (Subgraph, Subdigraph).

A subgraph of a graph G is a graph all of whose vertices are vertices of G and all of whose edges are edges of G.

A subdigraph of a digraph D is a digraph all of whose vertices are vertices of D and all of whose arcs are arcs of D.



Graph

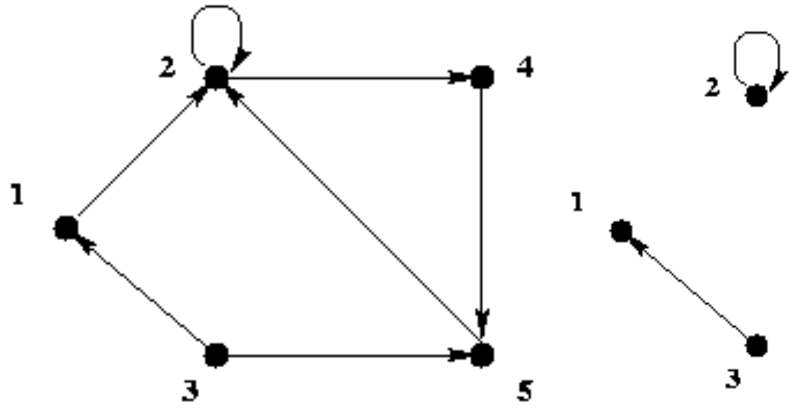


Subdigraph

Definition 2.10 (Partial Graph, Partial Digraph).

A partial graph of a graph G is a digraph consisting of arbitrary numbers of vertices and edges of G.

A partial digraph of a digraph D is a digraph consisting of arbitrary numbers of vertices and arcs of D.



Graph　　　Partial Digraph

Formal Mathematical Definition of a Subgraph

A graph $G' = (V', E')$ is a subgraph of another graph $G = (V, E)$ iff

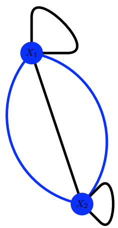
$V' \subseteq V$, and

$E' \subseteq E \wedge ((v1, v2) \in V \rightarrow (v1, v2) \in V')$

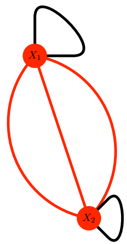
Note: In general, a subgraph need not have all possible edges.

Definition 2.11 (Induced Subgraph)**.**

If a subgraph has every possible edge, it is an induced subgraph.



Subgraph



Induced Subgraph

Definition 2.12 (Adjacency and incidence).

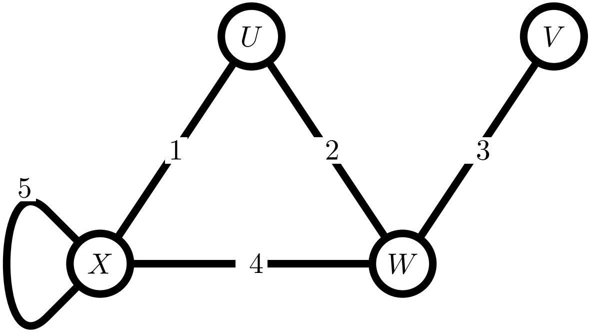
Two vertices v and w of a graph G are adjacent vertices if they are joined by an edge e.

The vertices v and w are then incident with the edge e and the edge e is incident with the vertices v and w.

Two vertices v and w of a digraph G are adjacent vertices if they are joined (in either direction) by an arc e.

An arc e that joins v to w is incident from v and incident to w.

Example 2.7 (Adjacency and incidence).



U and X are *adjacent*.

W is *incident* to $2, 3, 4$ and 5 is *incident* with X.

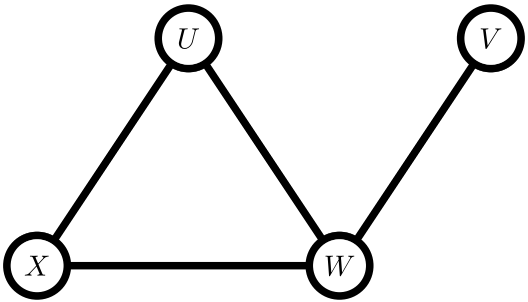
Back
Close

Definition 2.13 (Vertex Degree, Degree Sequence).

The degree of a vertex v is the number of edges incident with v, with each loop counted twice and is denoted by **deg** v.

The degree sequence of a graph G is the sequence obtained by listing the vertex degrees of G in descending order, with repeats as neccesary.

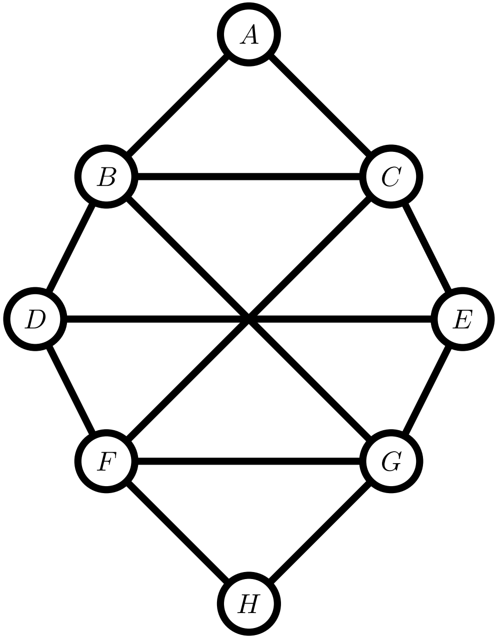
Example 2.8 (Vertex Degree, Degree Sequence).



The degree of U is 2
The degree of V is 1
The degree of W is 3
The degree of X is 2

So the degree sequence of the above graph is 3, 2, 2, 1

Problem 2.3 (Vertex Degree, Degree Sequence).

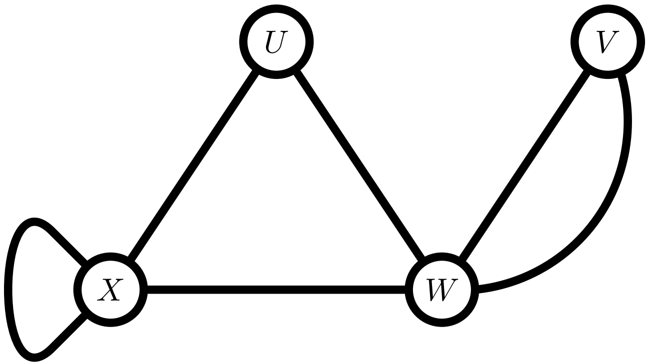


What are the degrees of the respective vertices $A, B, C, \ldots, H$?

What is the degree sequence of the above graph?

Definition 2.14 (Adjacency Matrix).

The *adjacency matrix*, *A, of a finite directed or undirected graph G with n vertices is the $n \times n$ matrix where the nondiagonal entry a_{ij} is the number of edges from vertex i to vertex j, and the diagonal entry a_{ii} the number of loops .*

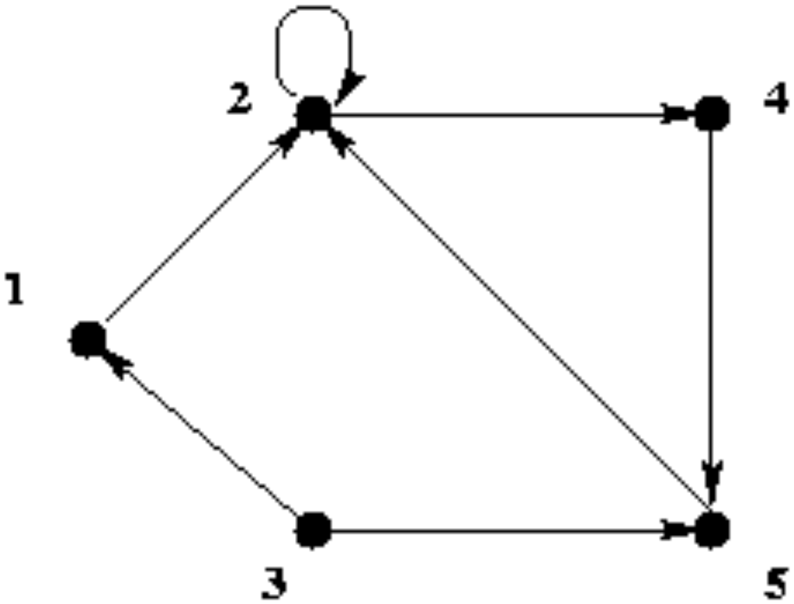


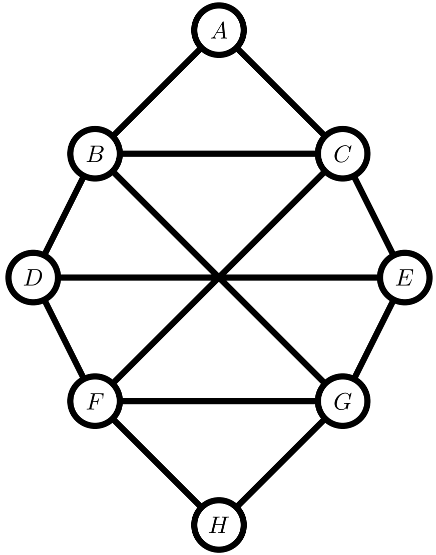
$$
\begin{array}{ll}
 & \overbrace{\begin{array}{cccc} U & V & W & X \end{array}}^{\text{Cols } 1\ldots n(=4)} \\
\begin{array}{ll} \text{(Row 1)} & U \\ \text{(Row 3)} & V \\ \text{(Row 3)} & W \\ \text{(Row 4)} & X \end{array} & \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 2 & 0 \\ 1 & 2 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}
\end{array}
$$

Back

Close

Problem 2.4 (Adjacency Matrix).





Write down the adjacency matrices of the above two graphs.

CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

Properties of an Adjacency Matrix

- There exists a unique adjacency matrix for each graph (up to permuting rows and columns), and it is not the adjacency matrix of any other graph.
- In the special case of a finite simple graph, the adjacency matrix is a (0,1)-matrix with zeros on its diagonal.
- If the graph is undirected, the adjacency matrix is symmetric.
- For sparse graphs, that is, graphs with few edges, an *adjacency list* is often preferred as a representation of the graph because it uses less space: *list of all edge (or arc) sets of vertices per edge (arc).*
- Another matrix representation for a graph is the *incidence matrix*: *a $p \times q$ matrix (B), where p and q are the numbers of vertices and edges respectively, such that $b_{ij} = 1$ if the vertex v_i and edge e_j are incident and 0 otherwise.*

Problem 2.5 (Properties of an Adjacency Matrix).

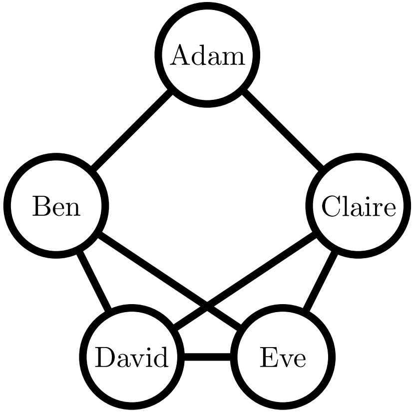
For each of the points on the previous slide write down a suitable graph and work out its adjacency matrix, adjecency list or incidence matrix.

Pay particular note to the size of each structure created

Lemma 2.15 (Handshaking lemma).
In any graph the, the sum of all vertex degrees is equal to twice the number of edges

Proof.
Each edge has two ends. □

The name handshaking lemma has its origin in the fact, that a group of people shaking hands can be described by a graph like



Here every vertex represents a person, and an edge appears as soon as those two people have shaken their hands.

Handshaking Lemma Corollaries

There are a few intuitive implications of the handshaking lemma:

- For a graph, the sum of degrees of all its nodes is even.
- In any graph, the sum of all the vertex-degrees is an even number.
- In any graph, the number of vertices of odd degree is even.
- If G is a graph which has n vertices and is regular of degree r, then G has exactly 1/2 nr edges.

Problem 2.6.

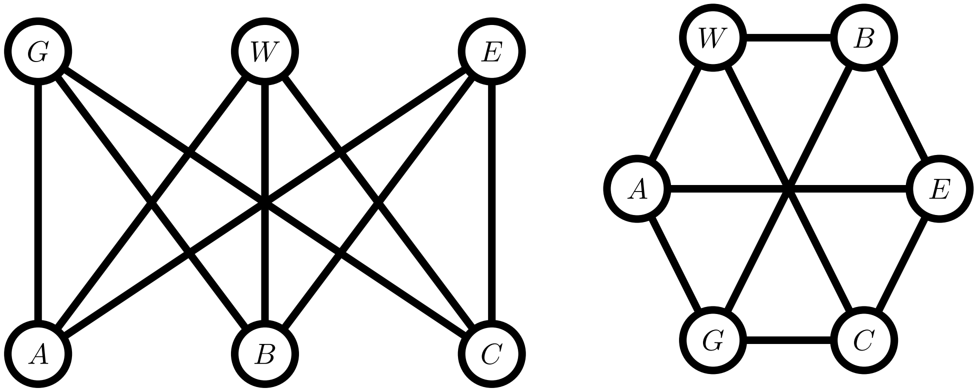
Prove the above corollaries.

The Similarity of Two Graphs (1)

It follows from our defintion of a graph, that it is completely determined by its edges and vertices.

This does not mean, that a graph can't be drawn in different ways.

For example the two graphs:



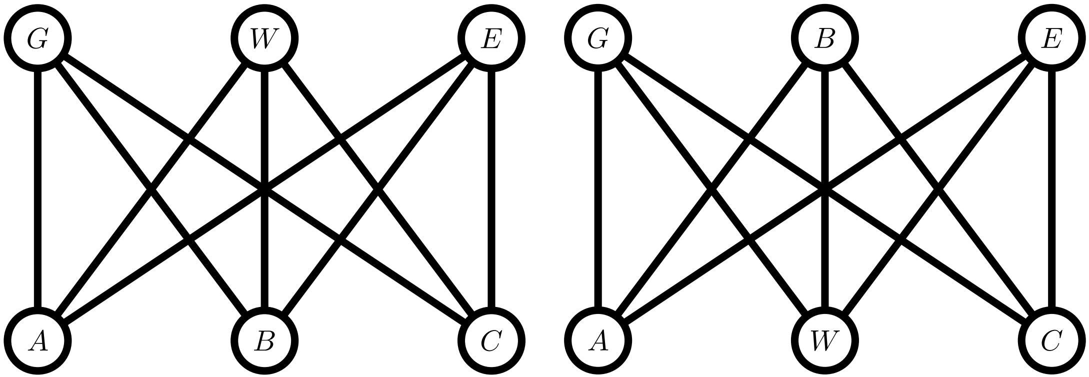
They look different at first sight, a closer look however reveals that, these are two pictures of the same graph.

Problem 2.7. *Write down and compare the adjacency matrices of the above graphs*

The Similarity of Two Graphs (2)

On the other hand, two graphs my look similar but represent different graphs.

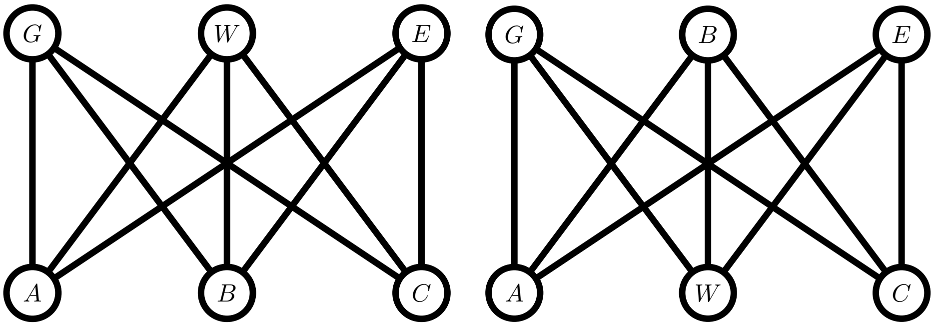
Consider the example:



Problem 2.8. *Write down and compare the adjacency matrices of the above graphs*

The Similarity of Two Graphs (3)

Continuing the example:



- AB is an edge of the second graph, but *not of the first one*.
- Although the graphs have essentially the same information they are not the same.
- However by relabelling the second graph, we can reproduce the first graph.

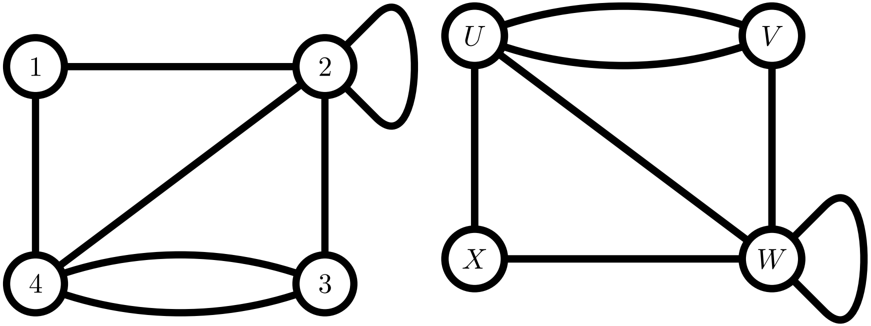
  Problem 2.9. *Which vertices should we relabel? and What Labels should they receive?*

- This leads to the following notion of ***Graph Isomorphisms***.

Definition 2.16 (Graph Isomorphism).

Two graphs G and H are isomorphic to each other, if H can be obtained by relabelling the vertices of G.
This means that there is a one-to-one correspondence between the vertices of G and H.
Such a one-to-one correspondence is called an isomorphism.

Example 2.9 (Graph Isomorphism).



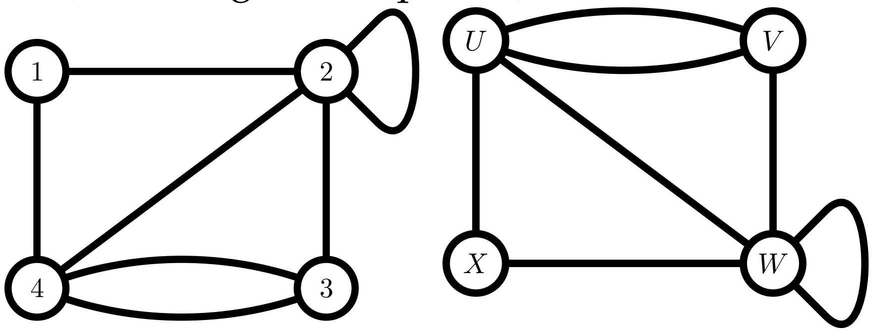
In the two graphs above show that their are isomorphic. *Which vertices correspond to each other?*

Checking Isomorphism

It is often hard to check whether two graphs are isomorphic or not. However we can give sufficient conditions for this.

- Two isomorphic graphs have the same degree sequence.
- Two graphs cannot be isomorphic if one of them contains a subgraph that the other does not.

Problem 2.10 (Checking Isomorphism).



Write out the degree sequence of the above graphs. Introduce a subgraph into one of the graphs and write out the new degree sequence

Paths and Cycles

Traversing a graph by travelling from one vertex to another is the ”bread and butter” of graph searching, sorting and optimisation algorithms.

This can readily become a non-trivial problem

Many problems can be posed as a graph travel problem.

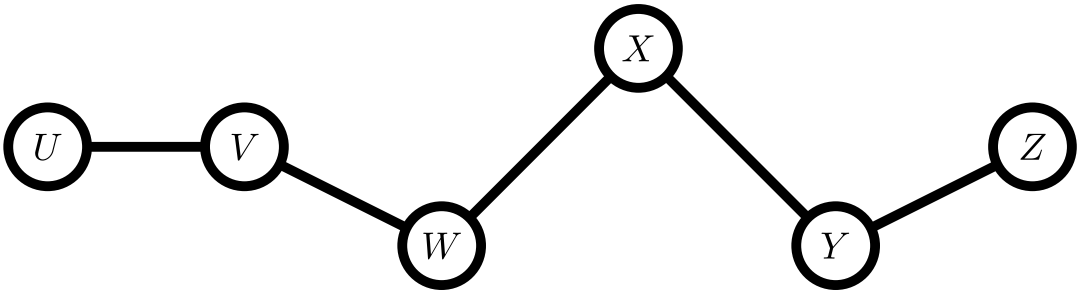
Many fancy algorithms have been designed over the years to address such problems.

Definition 2.17 (Walk).
A walk of lenght k in a graph G is a succession of k edges of the form

$$uv, vw, wx, \ldots, yz$$

This walk is denoted by $uvw \ldots z$, and is referred to as a walk ***between*** *u and z.*
A walk of length k in a digraph D is a succession of k arcs of the form $uv, vw, wx, \ldots, yz$. This walk is denoted by $uvw \ldots z$, and is referred to as a walk ***from*** *u and z.*

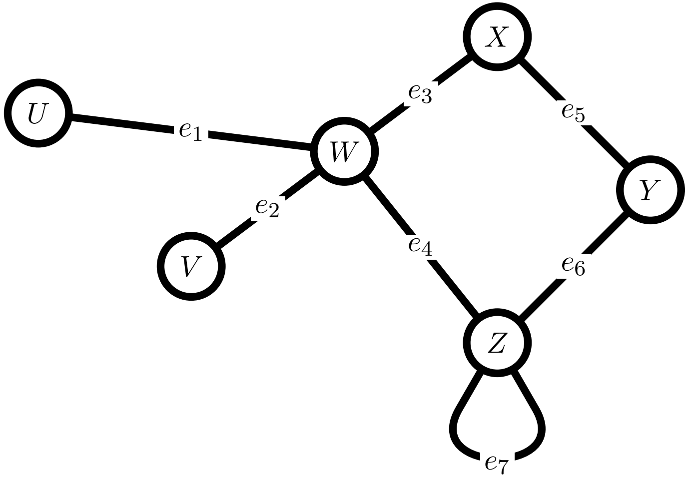
Example 2.10 (A simple walk).



The walk through this simple graph is: $UVWXYZ$

Example 2.11 (A more complex walk).

Note: *The definition of walk does not require that all edges or vertices in a walk are different:*



The most direct walk between U *and* Y *graph is:* $UWXY$

However a more roundabout walk could be: $UWVWZZY$

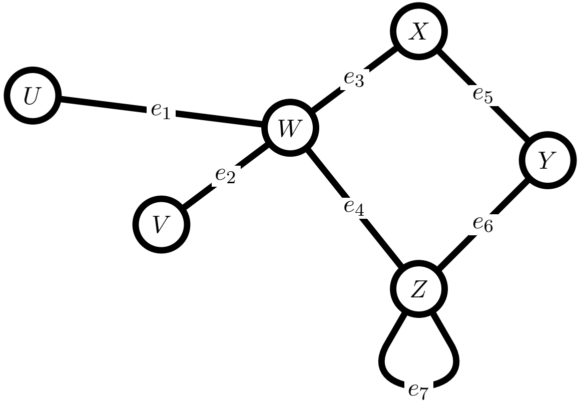
Back
Close

Alternative Graph Walk Notation

Notating such complex walks is more confusing

You can include edges and vertices in the walk list as:

$V_1 e_1 V_2 e_2 V_3 \ldots$ where V_i are consecutive vertices and e_i consecutive edges in the walk.



The last walk in the graph on the previous slide is more easily realised as:

$$U e_1 W e_2 V e_2 W e_4 Z e_7 Z e_6 Y$$

Definition 2.18 (Paths and trails).

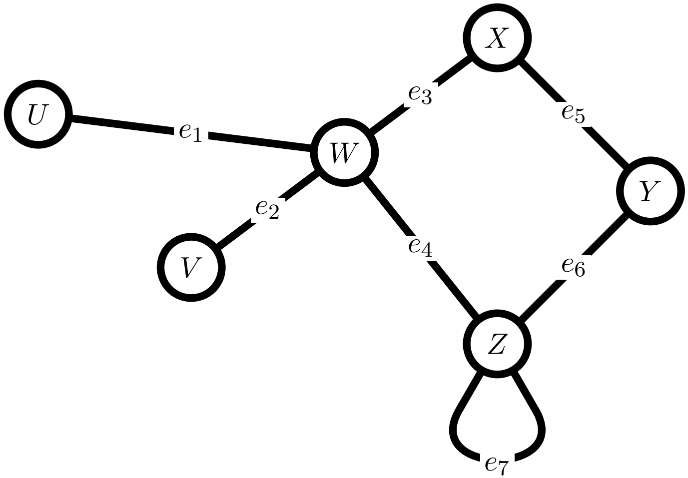
A trail in a graph G is a walk in which all the edges, but not necessarily all the vertices, are different.

A path in a graph G is a walk in which all the edges and all the vertices are different.

A trail in a digraph D is walk in which all the arcs, but not necessarily all the vertices, are different.

A path in a digraph D is a walk in which all the arcs and all the vertices are different.

Example 2.12 (Paths and trails).



The walk $Ue_1We_2Ve_2We_4Ze_7Ze_6Y$ *is* **not a trail or a path**

The walk $Ue_1We_4Ze_7Ze_6Y$ *is a* **a trail but not path**

The walk $Ue_1We_3Xe_5Ye_6Z$ *is a* **a path**

Definition 2.19 (Closed walks, paths and trails in graphs).
A closed walk in a graph G is a succession of edges of the form

$$uv, vw, wx, \ldots, yz, zu$$

that starts and ends at the same vertex.

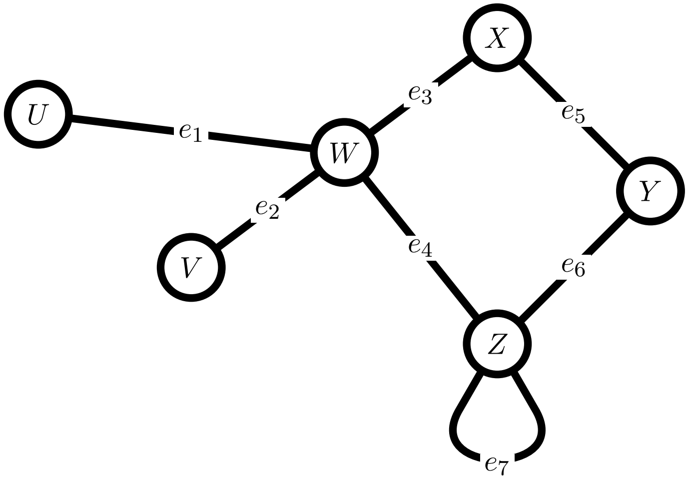
A closed trail in a graph G is a closed walk in G in which all the edges are different.

A cycle in a graph G is a closed walk in G in which all the edges are different and all theintermediate vertices are different.

A walk or trail is open if it starts and finishes at different vertices.

Definition 2.20 (Closed walks, paths and trails in digraphs). *The same definition as above is valid for digraphs with edges replaced by arcs.*

Example 2.13 (Closed walks, paths and trails in graphs).



The walk $Ue_1We_3Xe_5Ye_6Ze_4We_1U$ *is* **closed walk**

The walk $We_4Ze_7Ze_6Ye_5Xe_3W$ *is* **a closed trail**

The walk $We_3Xe_5Ye_6Ze_4W$ *is* **a cycle**

The walk $Ue_1We_4Ze_7Ze_6Y$ *is an* **open walk**

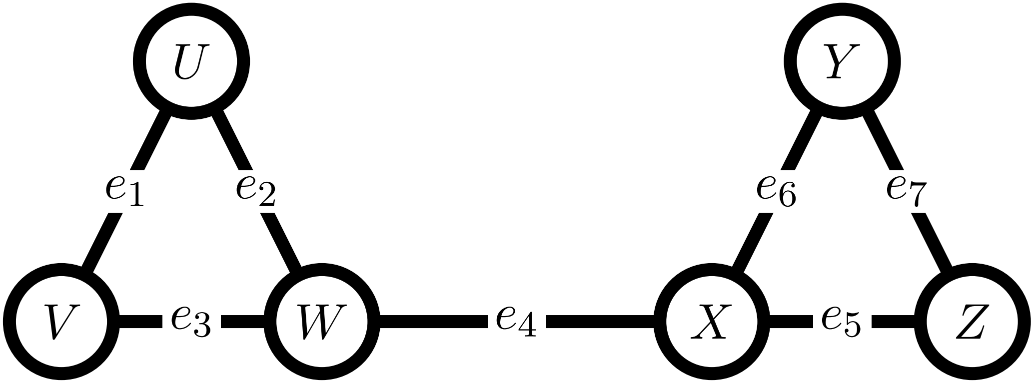
Definition 2.21 (Connectivity of a graph).

A graph G is connected if there is a path between each pair of vertices, and is disconnected otherwise.

An edge in a connected graph G is a bridge if its removal leaves a disconnected graph.

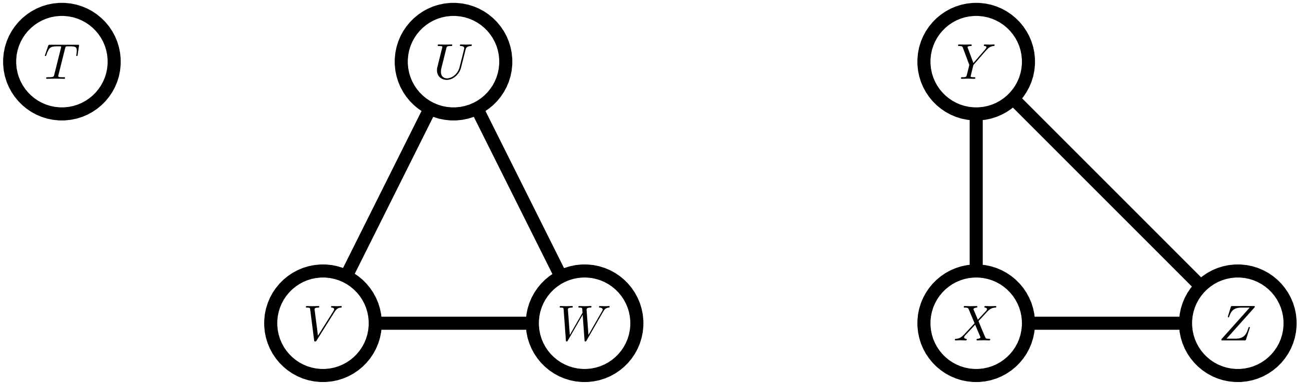
Every disconnected graph can be split up into a number of connected subgraphs, called components.

Example 2.14 (Connectivity of a graph: Bridge).



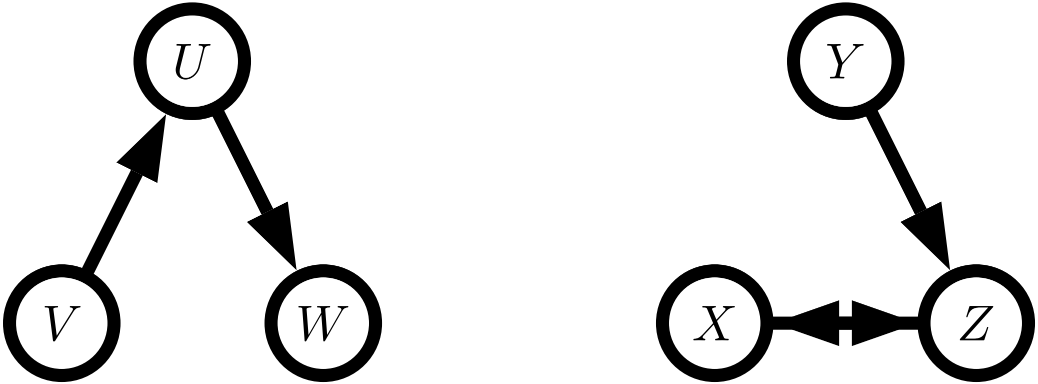
e_4 *is a* **bridge**.

Example 2.15 (Connectivity of a graph:Disconnections/Components).



A **disconnected graph** *with* **3 components**

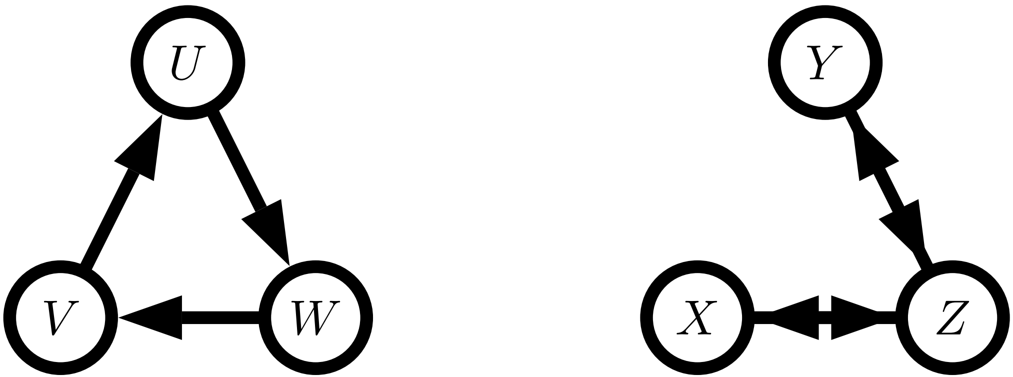
Definition 2.22 (Weak Connectivity of a digraph).



A digraph D is weakly connected if its underlying graph G is a connected graph and is disconnected otherwise.

That is to say if there is an undirected path between any pair of vertices.

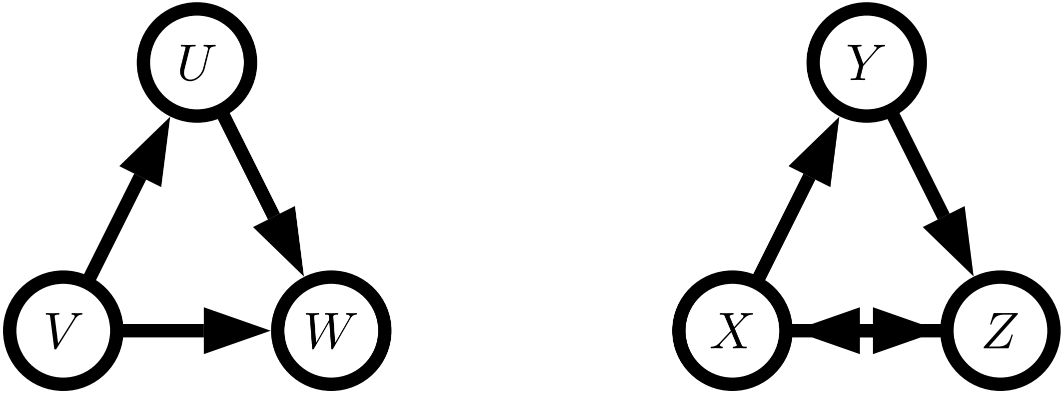
Definition 2.23 (Srong Connectivity of a digraph).



A digraph is strongly connected if there is a path between each pair of vertices.

That is to say it is possible to reach any node starting from any other node by traversing edges in the direction(s) in which they point.

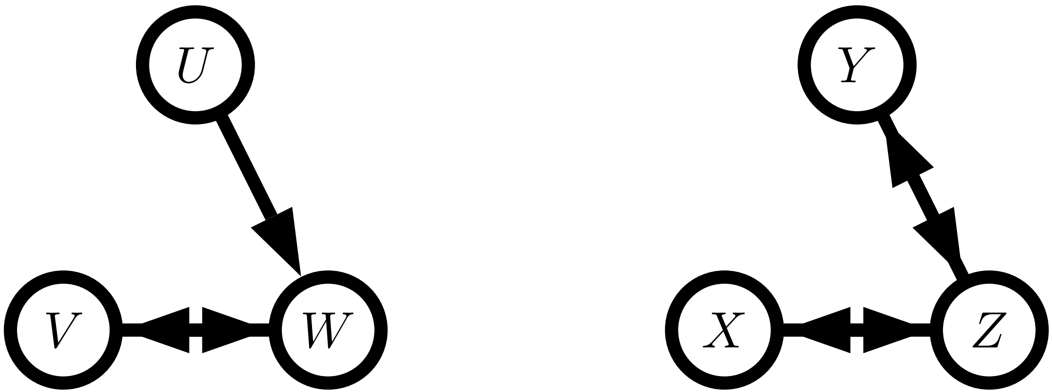
Problem 2.11 (Connectivity of a digraph).



Are the digraphs above weakly or strongly connected?

Justify your answer.

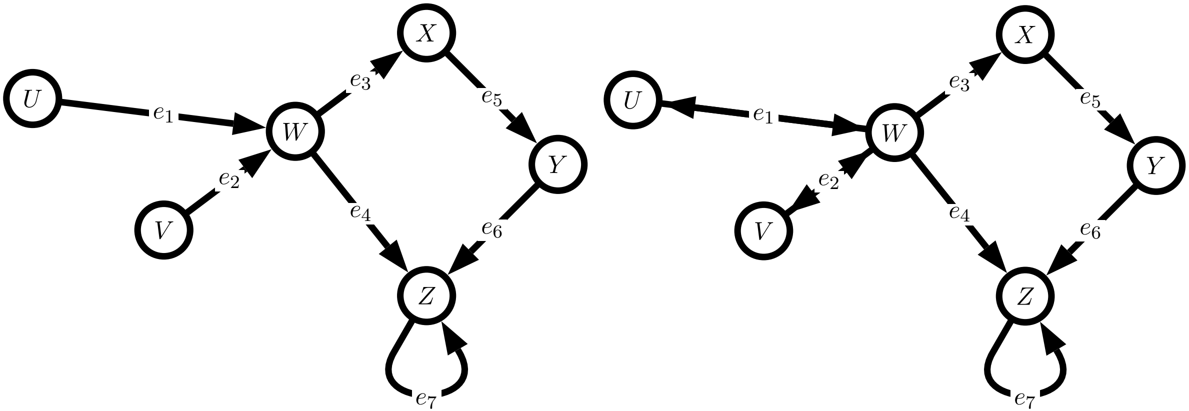
Problem 2.12 (Connectivity of a digraph).



Are the digraphs above weakly or strongly connected?

Justify your answer.

Problem 2.13 (Connectivity of a digraph).



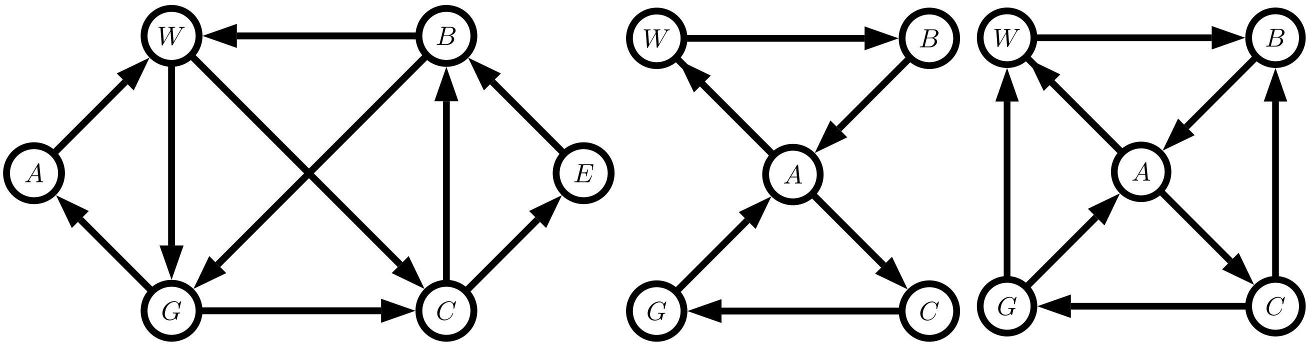
Are the digraphs above weakly or strongly connected?

Justify your answer.

Definition 2.24 (Eulerian trail).
A connected graph G is called Eulerian *if it contains a closed trail that includes* every edge*. Such a trail is called an* Eulerian trail*.*

Definition 2.25 (Hamiltonian Cycle).
A connected graph G is called Hamiltonian *if it contains a cycle that includes* every vertex*. Such a cycle is called a* Hamiltonian cycle*.*

Problem 2.14 (Eulerian trail/Hamiltonian Cycle).



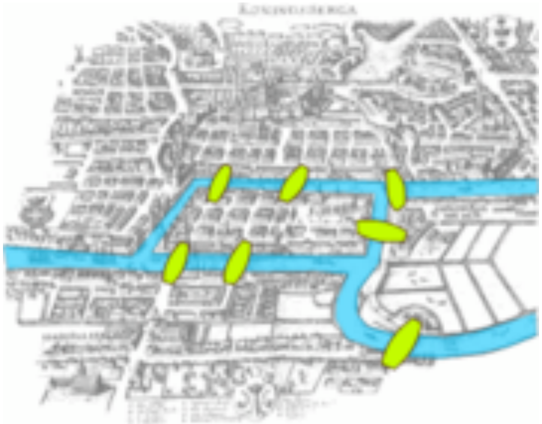
Do the above graphs contain Eulerian Trails and/or Hamiltonian Cycles?

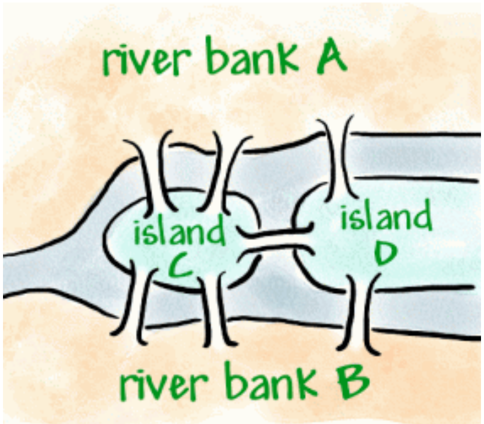
Example: The Königsberg bridge problem

The notion of an Eulerian trail goes back to Leonard Euler, who solved the so-called Königsberg bridge problem.

The Königsberg bridge problem was the following:

Is it possible to cross each of the seven bridges of Königsberg exactly once and return to the starting point?



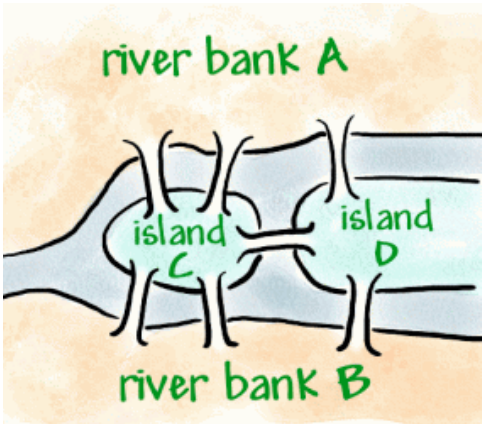


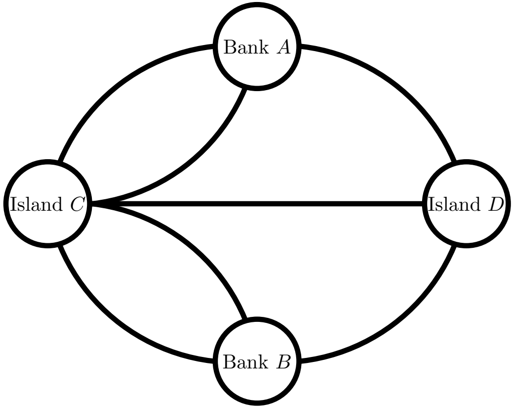
Euler's Solution

Leonard Euler solved this problem by the simple observation:

This would only be possible if whenever you cross into a part of the city you must be able to leave it by another bridge.

Rephrasing this problem in the language of graph theory, we get the problem of finding an **Eulerian trail** in the connected graph



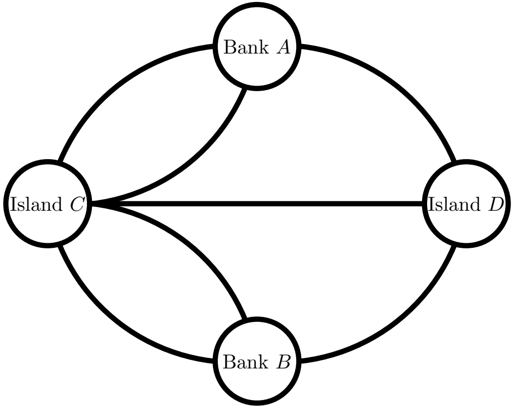


Euler's Theorem

The solution to this problem is given by the following theorem.

Theorem 2.26.
A connected graph is Eulerian if and only if each vertex has even degree.

Problem 2.15 (Königsberg bridge problem).



What are the degrees of the vertices in the graph of the Königsberg bridge problem?

Königsberg bridge problem: Solution

Since the degrees of all the vertices in the graph in the Königsberg bridge problem are **not even**:

The answer is that it is **not possible** to cross each of the seven bridges of Königsberg exactly once and return to the starting point.

Problem 2.16 (Königsberg bridge problem)**.**

The Königsberg bridge problem could have been solved if **one** *bridge was removed and another added.*
Which bridge would you remove and where would you add a bridge?