Announcements:

Next Monday public holiday.

3 Monday 6005 workshops

3 replacements + make-up workshop Friday afternoon

Attend any of the four options.

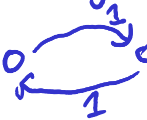
steady state:

$T S = S$

Recap:

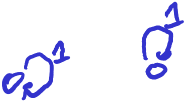
Markov processes.

steady states vs. long-run probabilities

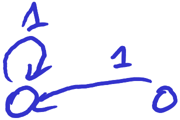


: has unique steady state.

Doesn't stabilise



: Every state is stationary



unique steady state

Any starting state tends to steady state

D1. Graph Theory

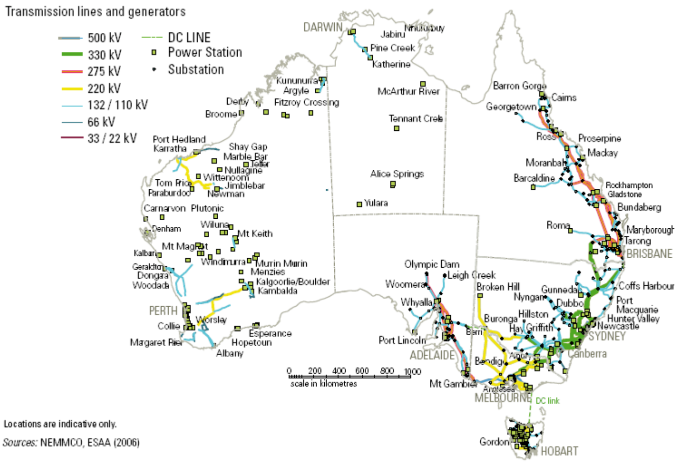
Notes originally prepared by Judy-anne Osborn.
Editing, expansion and additions by Malcolm Brooks.

Text Reference (Epp)	3ed:	Chapter 11
	4ed:	Chapter 10
	5ed:	Chapter 10

These references may not *completely* cover everything in this section, but they do have most it. They also contain a few items we do not cover.

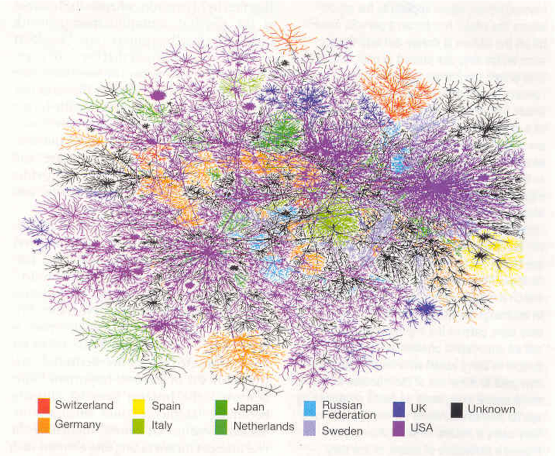
Real-world phenomena often modeled with graphs:

Australian Power Transmission Network

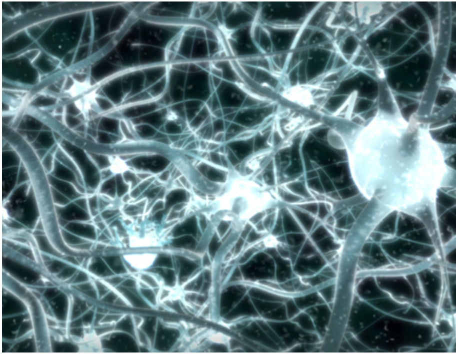


Complex Network Example: Internet

(William R. Cheswick)



Brain Network



from documentary, ‘Inside the living body’
http://abcnews.go.com/2020/popup?id=3560899

Graphs

A **graph** G is a collection of **vertices** and **edges**.

Graphs

A **graph** G is a collection of **vertices** and **edges**.

- The set of vertices of G is denoted $\boldsymbol{V(G)}$.

Graphs

A **graph** G is a collection of **vertices** and **edges**.

- The set of vertices of G is denoted $\boldsymbol{V(G)}$.
- The (multi)set[1] of edges of G is denoted $\boldsymbol{E(G)}$.

Graphs

A **graph** G is a collection of **vertices** and **edges**.

- The set of vertices of G is denoted $\boldsymbol{V(G)}$.
- The (multi)set[1] of edges of G is denoted $\boldsymbol{E(G)}$.

[1] As explained in Section C1, a 'multiset' is just like a set except that it may contain the same element more than once.

Graphs

A **graph** G is a collection of **vertices** and **edges**.

- The set of vertices of G is denoted $\boldsymbol{V(G)}$.
- The (multi)set[1] of edges of G is denoted $\boldsymbol{E(G)}$.
- Each edge is specified by a pair of vertices.
  (The two vertices could be the same.)

[1]As explained in Section C1, a 'multiset' is just like a set except that it may contain the same element more than once.

Graphs

A **graph** G is a collection of **vertices** and **edges**.

- The set of vertices of G is denoted $\boldsymbol{V(G)}$.
- The (multi)set[1] of edges of G is denoted $\boldsymbol{E(G)}$.
- Each edge is specified by a pair of vertices.
  (The two vertices could be the same.)
- In this course we only consider **finite** graphs;
  i.e. $V(G)$ and $E(G)$ are both finite sets.

[1] As explained in Section C1, a 'multiset' is just like a set except that it may contain the same element more than once.

Diagrams of Graphs

To draw a graph we use:

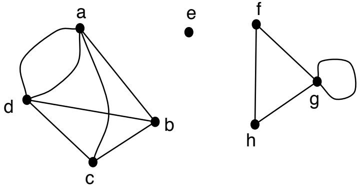
- dots/circles for vertices
- lines for edges

Diagrams of Graphs

To draw a graph we use:

- dots/circles for vertices
- lines for edges

Example: The graph G

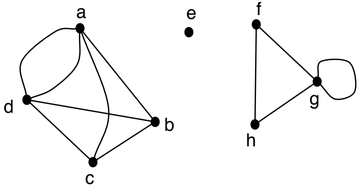


Diagrams of Graphs

To draw a graph we use:

- dots/circles for vertices
- lines for edges

Example: The graph G



has vertex set $V(G) = \{a, b, c, d, e, f, g, h\}$
and edge multiset $E(G) =$
$\{\{a,b\}, \{a,c\}, \{a,d\}, \{a,d\}, \{b,c\}, \{b,d\}, \{c,d\}, \{f,g\}, \{f,h\}, \{g,g\}, \{g,h\}\}.$

A table of edges

- The same graph as a table of labelled edges:

Edge	Endpoints
e_1	$\{a, b\}$
e_2	$\{a, c\}$
e_3	$\{a, d\}$
e_4	$\{a, d\}$
e_5	$\{b, c\}$
e_6	$\{b, d\}$

Edge	Endpoints
e_7	$\{c, d\}$
e_8	$\{f, g\}$
e_9	$\{f, h\}$
e_{10}	$\{g, g\}$
e_{11}	$\{g, h\}$

A table of edges

- The same graph as a table of labelled edges:

Edge	Endpoints
e_1	$\{a, b\}$
e_2	$\{a, c\}$
e_3	$\{a, d\}$
e_4	$\{a, d\}$
e_5	$\{b, c\}$
e_6	$\{b, d\}$

Edge	Endpoints
e_7	$\{c, d\}$
e_8	$\{f, g\}$
e_9	$\{f, h\}$
e_{10}	$\{g, g\}$
e_{11}	$\{g, h\}$

- Notice that e_3 is distinct from e_4 even though both edges have the same endpoints.

A vertex adjacency listing

- The same graph as a vertex adjacency listing:

Vertex	Adjacent to:
a	b, c, d, d
b	a, c, d
c	a, b, d
d	a, a, b, c
e	
f	g, h
g	f, g, h
h	f, g

An adjacency matrix

- An adjacency matrix for the same graph

$$
\begin{array}{c@{\;}c}
 & \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\
\begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{array} &
\begin{bmatrix}
0 & 1 & 1 & \underline{2} & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
\underline{2} & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0
\end{bmatrix}
\end{array}
$$

no change, because edges are undirected.

An adjacency matrix

- An adjacency matrix for the same graph

$$\begin{array}{c|c} & \begin{matrix} a & b & c & d & e & f & g & h \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{matrix} & \begin{bmatrix} 0 & 1 & 1 & 2 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

- The $(i, j)^{\text{th}}$ entry is the number of edges between vertices i and j.

An adjacency matrix

- An adjacency matrix for the same graph

$$
\begin{array}{c@{\;}c}
 & \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\
\begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{array} &
\begin{bmatrix}
0 & 1 & 1 & 2 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0
\end{bmatrix}
\end{array}
$$

- The $(i, j)^{\text{th}}$ entry is the number of edges between vertices i and j.
- Thus $a_{i,j}$ is number of ways that i is adjacent to j.

Some Graph Terminology

- An edge connects its **endpoints**.

Some Graph Terminology

- An edge connects its **endpoints**.
- An edge with both endpoints the same is called a **loop**.

Some Graph Terminology

- An edge connects its **endpoints**.
- An edge with both endpoints the same is called a **loop**.
- Two edges may connect the same pair of endpoints, in which case they are said to be **parallel**.

Some Graph Terminology

- An edge connects its **endpoints**.
- An edge with both endpoints the same is called a **loop**.
- Two edges may connect the same pair of endpoints, in which case they are said to be **parallel**.
- Two vertices are **adjacent** if they are connected by an edge; two edges are **adjacent** if they share an endpoint.

Some Graph Terminology

- An edge is **incident on** its endpoints.

Some Graph Terminology

- An edge is **incident on** its endpoints.
- A vertex with no incident edges is **isolated**.

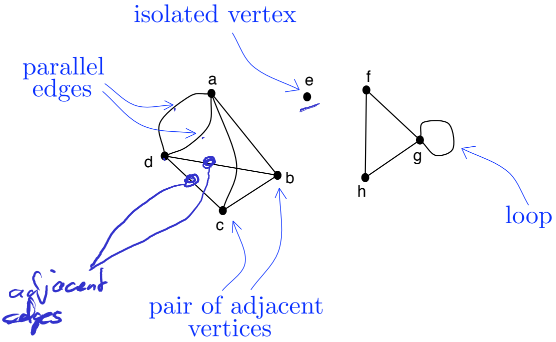
Some Graph Terminology

- An edge is **incident on** its endpoints.
- A vertex with no incident edges is **isolated**.
- A graph with no vertices (hence no edges) is **empty**.

Some Graph Terminology

- An edge is **incident on** its endpoints.
- A vertex with no incident edges is **isolated**.
- A graph with no vertices (hence no edges) is **empty**.
- The **order** of a graph, G, is the number of vertices in it, i.e. $|V(G)|$.
  (A graph of order ‘0’ is empty.)

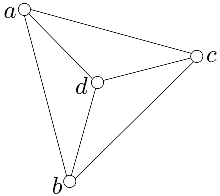
Some graph concepts illustrated



Another example

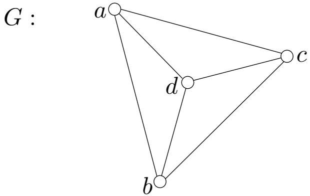
- Tetrahedron Graph

$G:$

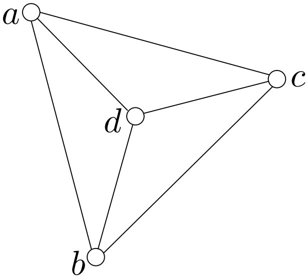


Another example

- Tetrahedron Graph



  - $V(G) = \{a, b, c, d\}$
  - $E(G) = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}\}$



- Tetrahedron Graph:

Adjacency listing:

Vertex	Adjacent to:
a	b, c, d
b	a, c, d
c	a, b, d
d	a, b, c

Adjacency matrix:

$$\begin{array}{c c} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

More about graph diagrams

- Position, length, curvedness and orientation in a graph diagram do not matter for the graph represented.

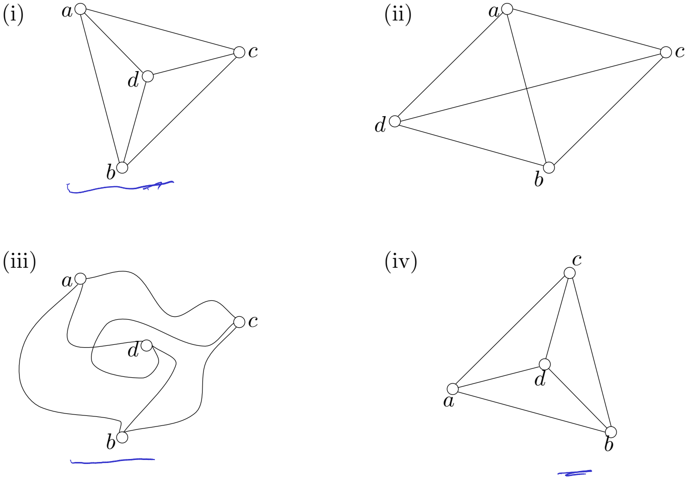
More about graph diagrams

- Position, length, curvedness and orientation in a graph diagram do not matter for the graph represented.
- The only things which matter are that precisely those vertices in $V(G)$ are shown and precisely those edges in $E(G)$ are shown.

More about graph diagrams

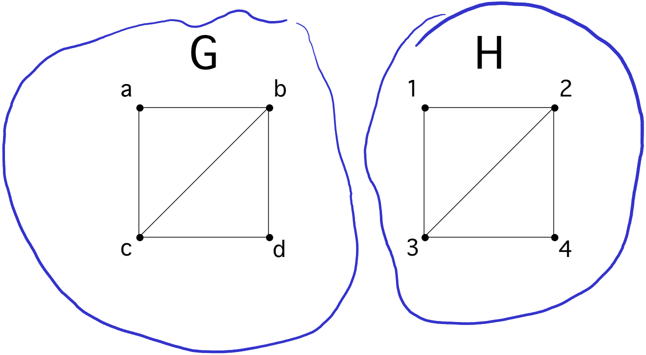
- Position, length, curvedness and orientation in a graph diagram do not matter for the graph represented.
- The only things which matter are that precisely those vertices in $V(G)$ are shown and precisely those edges in $E(G)$ are shown.
- For instance, the following diagrams all represent the same graph.

Four diagrams of the same graph



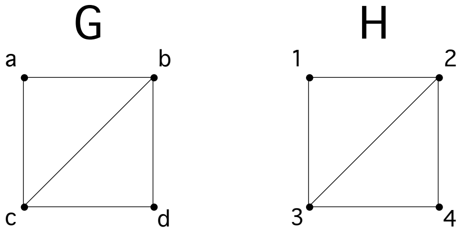
Isomorphic Graphs

Consider graphs G and H as pictured:



Isomorphic Graphs

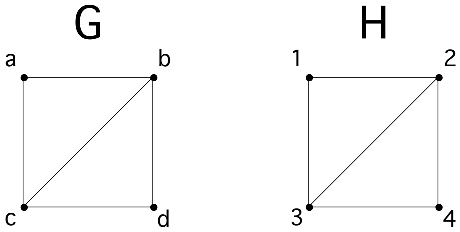
Consider graphs G and H as pictured:



- They are different graphs because their vertex labels are different.

Isomorphic Graphs

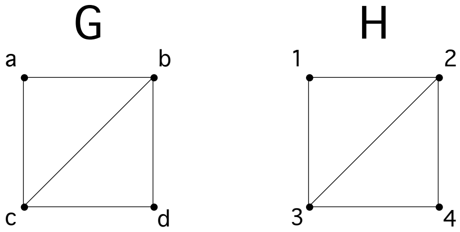
Consider graphs G and H as pictured:



- They are different graphs because their vertex labels are different.
- But they are the same in some sense.

Isomorphic Graphs

Consider graphs G and H as pictured:



- They are different graphs because their vertex labels are different.
- But they are the same in some sense.
- Formally these graphs are called ‘isomorphic’.

Isomorphisms

An **isomorphism** between two graphs G_1 and G_2 is a bijection

$$f : V(G_1) \to V(G_2)$$

such that:

Isomorphisms

An **isomorphism** between two graphs G_1 and G_2 is a bijection

$$f : V(G_1) \to V(G_2)$$

such that:

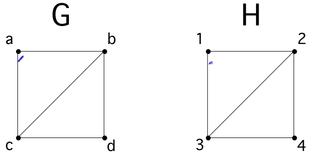
$\{u, v\}$ is an edge in $E(G_1)$

if and only if

with multiplicity

$\{f(u), f(v)\}$ is an edge in $E(G_2)$

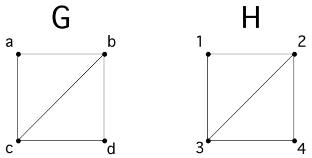
Isomorphisms



An example of an isomorphism between G and H is the mapping

$$f : V(G) \rightarrow V(H)$$

Isomorphisms

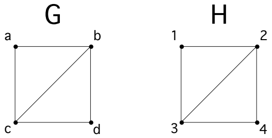


An example of an isomorphism between G and H is the mapping

$$f : V(G) \rightarrow V(H)$$

$$\begin{aligned} a &\mapsto 1 \\ b &\mapsto 2 \\ c &\mapsto 3 \\ d &\mapsto 4 \end{aligned}$$

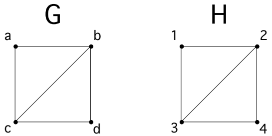
Isomorphisms



The mapping ‘preserves’ edges:

$$\begin{array}{cccc}
\{a,b\} & \mapsto & \{1,2\} & \checkmark \\
\{a,c\} & \mapsto & \{1,3\} & \checkmark \\
\{b,c\} & \mapsto & \{2,3\} & \checkmark \\
\{b,d\} & \mapsto & \{2,4\} & \checkmark \\
\{c,d\} & \mapsto & \{3,4\} & \checkmark
\end{array}$$

Isomorphisms



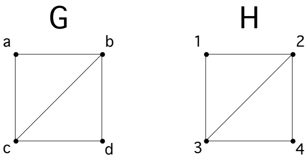
The mapping 'preserves' edges:

$$\begin{array}{cccc} \{a,b\} & \mapsto & \{1,2\} & \checkmark \\ \{a,c\} & \mapsto & \{1,3\} & \checkmark \\ \{b,c\} & \mapsto & \{2,3\} & \checkmark \\ \{b,d\} & \mapsto & \{2,4\} & \checkmark \\ \{c,d\} & \mapsto & \{3,4\} & \checkmark \end{array}$$

....and non-edges:

$$\{a,d\} \mapsto \{1,4\} \quad \checkmark$$

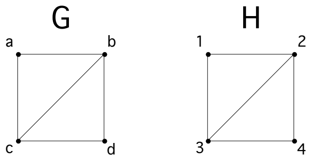
Isomorphisms



A different example of an isomorphism between G and H is the mapping

$$g : V(G) \quad \rightarrow \quad V(H)$$

Isomorphisms



A different example of an isomorphism between G and H is the mapping

$$g : V(G) \rightarrow V(H)$$

$$\begin{aligned} a &\mapsto 1 \\ b &\mapsto 3 \\ c &\mapsto 2 \\ d &\mapsto 4 \end{aligned}$$

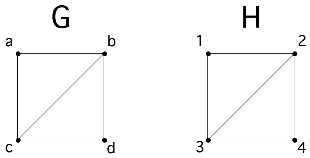
Isomorphic Graphs

If there exists an isomorphism between two graphs then the graphs are said to be **isomorphic**.

Isomorphic Graphs

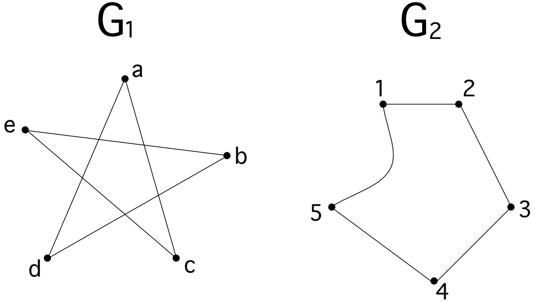
If there exists an isomorphism between two graphs then the graphs are said to be **isomorphic**.

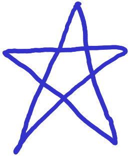
Example: Graphs G and H are isomorphic.



Isomorphic Graphs

- The following graphs pictured are isomorphic.

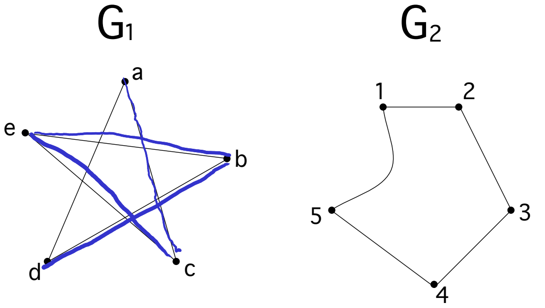




Isomorphic Graphs

$a \mapsto 1$
$c \mapsto 2$
$e \mapsto 3$
$b \mapsto 4$
$d \mapsto 5$

- The following graphs pictured are isomorphic.



- Can you specify an explicit isomorphism between them?

Directed Graphs

Digraphs

Digraphs were introduced in Section A3 in order to represent some relations diagrammatically. Here we look at digraphs in general.

Digraphs

Digraphs were introduced in Section A3 in order to represent some relations diagrammatically. Here we look at digraphs in general.

- A **directed graph** (or **digraph**) is the same as a graph except that edges are *ordered* pairs of endpoints.

Digraphs

Digraphs were introduced in Section A3 in order to represent some relations diagrammatically. Here we look at digraphs in general.

- A **directed graph** (or **digraph**) is the same as a graph except that edges are *ordered* pairs of endpoints.
- Each edge has an **initial vertex** and a terminal **final vertex**.

Digraphs

Digraphs were introduced in Section A3 in order to represent some relations diagrammatically. Here we look at digraphs in general.

- A **directed graph** (or **digraph**) is the same as a graph except that edges are *ordered* pairs of endpoints.

- Each edge has an **initial vertex** and a **final vertex**.

- Loops are still allowed.

Digraphs

Digraphs were introduced in Section A3 in order to represent some relations diagrammatically. Here we look at digraphs in general.

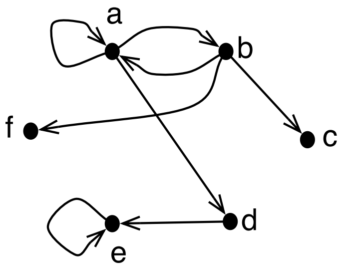
- A **directed graph** (or **digraph**) is the same as a graph except that edges are *ordered* pairs of endpoints.
- Each edge has an **initial vertex** and a **final vertex**.
- Loops are still allowed.
- The edges of a digraph are sometimes called **arcs**.

Digraphs

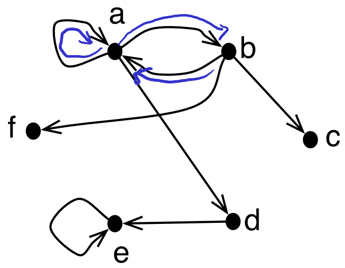
Digraphs were introduced in Section A3 in order to represent some relations diagrammatically. Here we look at digraphs in general.

- A **directed graph** (or **digraph**) is the same as a graph except that edges are *ordered* pairs of endpoints.
- Each edge has an **initial vertex** and a **final vertex**.
- Loops are still allowed.
- The edges of a digraph are sometimes called **arcs**.
- In a diagram of a digraph, the direction of an arc is given by an arrow.

Example



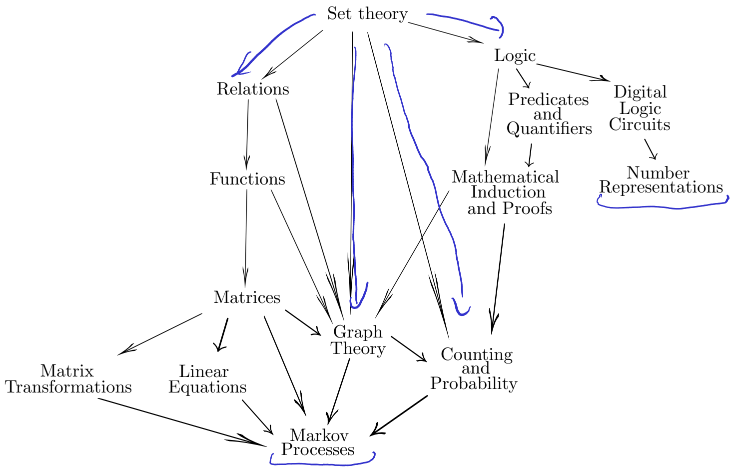
Example



The vertex set and edge set for this graph are:

$$
\begin{aligned}
V(G) &= \{a, b, c, d, e, f\} \\
E(G) &= \{(a,a), (a,b), (a,d), (b,a), (b,c), (b,f), (d,e), (e,e)\}
\end{aligned}
$$

An application: Recording Information Dependencies



An arrow from A to B means that B depends upon A in some way.

Foodwebs

- An application of digraphs in ecology is in describing a *foodweb*.

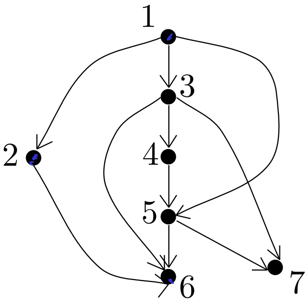
Foodwebs

- An application of digraphs in ecology is in describing a *foodweb*.
- The next slide shows a foodweb developed by Parsons and LeBrasseur, as adapted by Cohen, pertaining to the following species in the Strait of Georgia, British Columbia.

KEY	SPECIES
1.	Juvenile pink salmon
2.	P. Minutus
3.	Calanusand Euphausiid Burcillia
4.	Euphausiid Eggs
5.	Euphausiids
6.	Chaetoceros Socialis and Debilis
7.	Mu-Flagellates

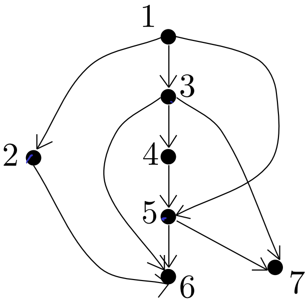
Example

An arrow from i to j means *'i eats j'*:



Example

An arrow from i to j
means *'i eats j'* :

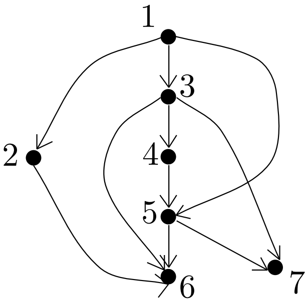


For example:

- species 1 eats species 2, 3, and 5

Example

An arrow from i to j means *'i eats j'* :

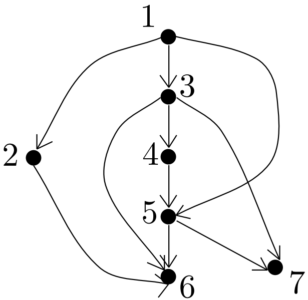


For example:

- species 1 eats species 2, 3, and 5
- species 4 *only* eats species 5.

Example

An arrow from i to j
means 'i eats j' :



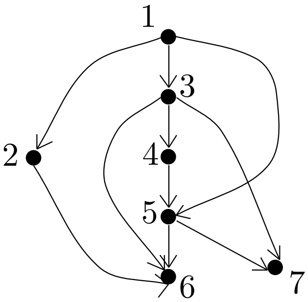
For example:

- species 1 eats species 2, 3, and 5
- species 4 *only* eats species 5.

Note: Some foodweb diagrams have their arrows *reversed*:
i.e. an arrow from A to B means 'A is food for B'.

Example

An arrow from i to j
means 'i *eats* j':



For example:

- species 1 eats species 2, 3, and 5
- species 4 *only* eats species 5.

Note: Some foodweb diagrams have their arrows *reversed*:
i.e. an arrow from A to B means 'A is food for B'.

We shall use the first convention unless stated otherwise.

Niche Overlap Graphs

- An application of graphs in ecology is in describing commonalities (or competition) between species in a *Niche Overlap Graph*.

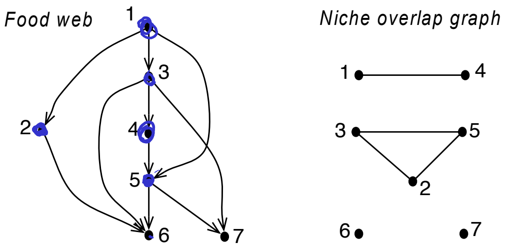
Niche Overlap Graphs

- An application of graphs in ecology is in describing commonalities (or competition) between species in a *Niche Overlap Graph*.
- Each species is represented by a vertex. An undirected edge connects two vertices if and only if the species represented by these vertices compete for food.

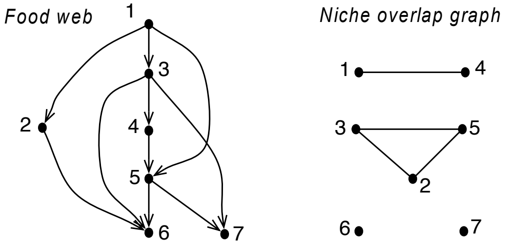
Niche Overlap Graphs

- An application of graphs in ecology is in describing commonalities (or competition) between species in a *Niche Overlap Graph*.
- Each species is represented by a vertex. An undirected edge connects two vertices if and only if the species represented by these vertices compete for food.
- The following niche overlap graph is constructed from the Food Web data of the previous example.

Food Webs and Niche Overlap Graphs



Food Webs and Niche Overlap Graphs



- For example:
  species 1 and 4 compete for food (species 5), so are connected by an edge in the niche overlap graph.

Types of Graphs and Digraphs

Sometimes it is useful to restrict our attention to two types of graphs and digraphs, called:

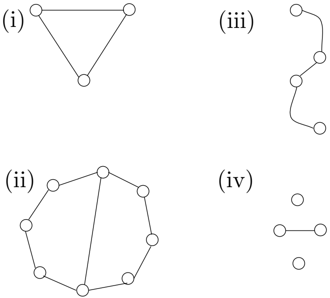
- **Simple Graphs** and
- **Simple Digraphs**

Simple Graphs

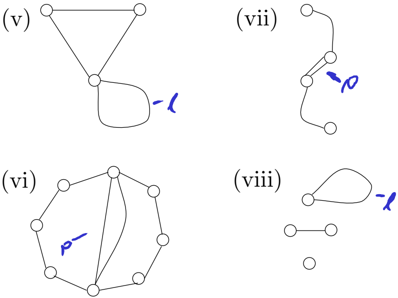
A **simple graph** is a graph that has no loops and no parallel edges.

Simple Graphs

A **simple graph** is a graph that has no loops and no parallel edges.



Some simple Graphs



Some non-simple Graphs

Terminology warning

- Warning: in graph theory, different authors use the *same words* to mean *different things*.

Terminology warning

- Warning: in graph theory, different authors use the *same words* to mean *different things*.
- For some, what we are calling a simple graph is just a graph.

Terminology warning

- Warning: in graph theory, different authors use the *same words* to mean *different things*.
- For some, what we are calling a simple graph is just a graph.
- For some, what we would call a graph with parallel edges, is a multi-graph.

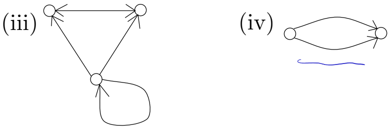
Simple Digraphs

Similarly, a **simple digraph** is a digraph that has no loops and no parallel edges.

Simple Digraphs

Similarly, a **simple digraph** is a digraph that has no loops and no parallel edges.

Some simple Digraphs



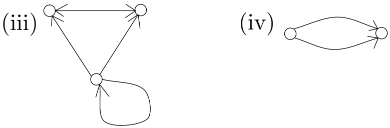
Some non-simple Digraphs

Don't allow duplicate ordered pairs in our edge set

Simple Digraphs

Similarly, a **simple digraph** is a digraph that has no loops and no parallel edges.

Some simple Digraphs



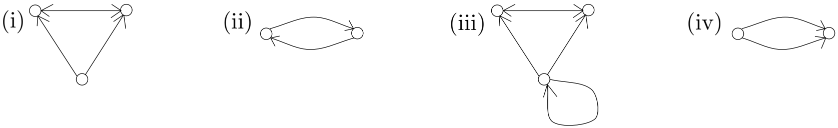
Some non-simple Digraphs

Note that:

- It is okay to have both (a, b) and (b, a) - these are not parallel;

Simple Digraphs

Similarly, a **simple digraph** is a digraph that has no loops and no parallel edges.



Some simple Digraphs

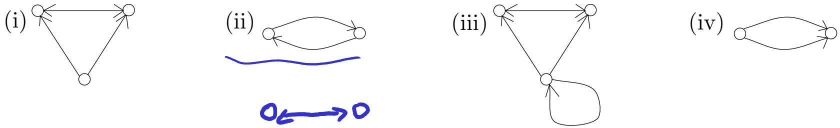
Some non-simple Digraphs

Note that:

- It is okay to have both (a, b) and (b, a) - these are not parallel;
- It is not okay to have (a, b) twice - those would be parallel.

Simple Digraphs

Similarly, a **simple digraph** is a digraph that has no loops and no parallel edges.



Some simple Digraphs *Some non-simple Digraphs*

Note that:

- It is okay to have both (a, b) and (b, a) - these are not parallel;
- It is not okay to have (a, b) twice - those would be parallel.
- We sometimes draw a single edge with an arrow at each end to indicate a pair of edges (a, b) and (b, a), instead of drawing two distinct lines.

Special simple graphs I: Complete Graphs

A **complete graph** on n vertices is a simple graph in which each pair of distinct vertices are adjacent (*i.e.* are 'joined' by an edge).

Special simple graphs I: Complete Graphs

A **complete graph** on n vertices is a simple graph in which each pair of distinct vertices are adjacent (*i.e.* are 'joined' by an edge).

A complete graph on n vertices is denoted by $\boldsymbol{K_n}$.

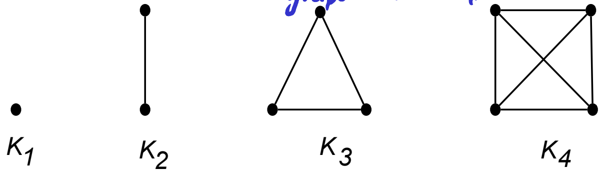
Special simple graphs I: Complete Graphs

A **complete graph** on n vertices is a simple graph in which each pair of distinct vertices are adjacent (*i.e.* are 'joined' by an edge).

A complete graph on n vertices is denoted by $\boldsymbol{K_n}$.

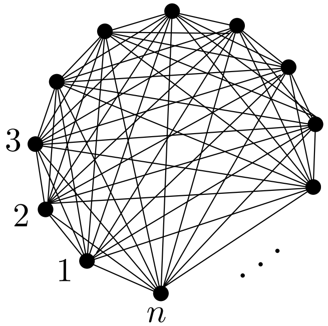
If G, H have n vertices and are complete, then any bijection $f : G \to H$ is going to preserve edges so is a graph isomorphism

Examples:



How many edges in K_n?

How many edges are there in K_n, the complete graph of order n?

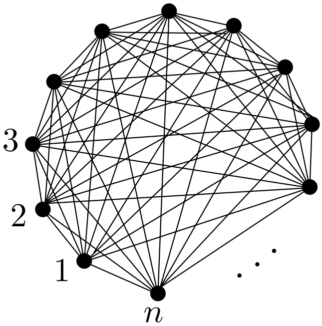


$\binom{n}{2}$

since we need an edge for every pair of vertices.

How many edges in K_n?

How many edges are there in K_n, the complete graph of order n?



The answer is $\boxed{\binom{n}{2} = \frac{n(n-1)}{2}}$. ~~Why?~~

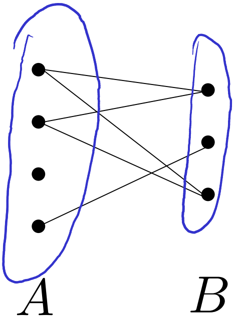
Special simple graphs II: Bipartite Graphs

A **bipartite** graph is a simple graph whose vertices can be partitioned into two disjoint sets A and B such that **every edge** of the graph connects a vertex in A to a vertex in B.

Special simple graphs II: Bipartite Graphs

A **bipartite** graph is a simple graph whose vertices can be partitioned into two disjoint sets A and B such that **every edge** of the graph connects a vertex in A to a vertex in B.

Example:



A larger example of a bipartite graph

Sometimes it is not obvious at first glance that a graph is bipartite.

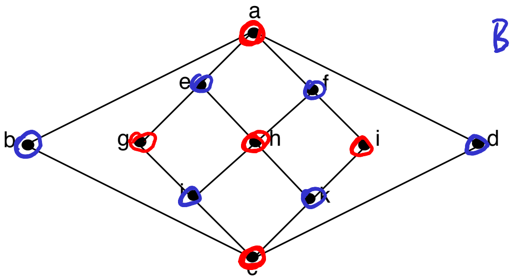
A larger example of a bipartite graph

Sometimes it is not obvious at first glance that a graph is bipartite.

Example: This graph is bipartite:

A =

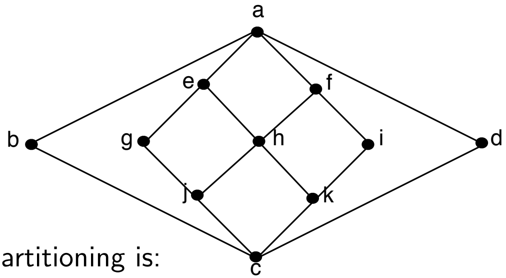
B =



A larger example of a bipartite graph

Sometimes it is not obvious at first glance that a graph is bipartite.

Example: This graph is bipartite:



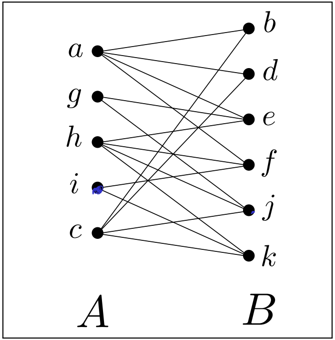
The vertex partitioning is:

$$\boxed{\begin{aligned} A &= \{a, g, h, i, c\} \\ B &= \{b, d, e, f, j, k\} \end{aligned}}$$

Larger example continued

planar graphs
are all 4-partite
4 colourable

This is the same graph, redrawn.



Special simple graphs III: Complete Bipartite Graphs

A **complete bipartite** graph is a simple graph whose vertices may be partitioned into two sets A and B such that:

Special simple graphs III: Complete Bipartite Graphs

A **complete bipartite** graph is a simple graph whose vertices may be partitioned into two sets A and B such that:

- $\forall a \in A$ and $\forall b \in B$, the edge $\{a, b\}$ belongs to the graph.

Special simple graphs III: Complete Bipartite Graphs

A **complete bipartite** graph is a simple graph whose vertices may be partitioned into two sets A and B such that:

- $\forall a \in A$ and $\forall b \in B$, the edge $\{a, b\}$ belongs to the graph.
- There are no other edges in the graph.

Special simple graphs III: Complete Bipartite Graphs

A **complete bipartite** graph is a simple graph whose vertices may be partitioned into two sets A and B such that:

- $\forall a \in A$ and $\forall b \in B$, the edge $\{a, b\}$ belongs to the graph.
- There are no other edges in the graph.

If A has m vertices and B has n vertices, the complete bipartite graph on A and B is denoted by $\boldsymbol{K_{m,n}}$.

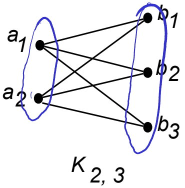
Special simple graphs III: Complete Bipartite Graphs

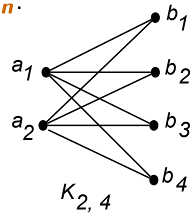
A **complete bipartite** graph is a simple graph whose vertices may be partitioned into two sets A and B such that:

- $\forall a \in A$ and $\forall b \in B$, the edge $\{a, b\}$ belongs to the graph.
- There are no other edges in the graph.

If A has m vertices and B has n vertices, the complete bipartite graph on A and B is denoted by $\boldsymbol{K_{m,n}}$.

Examples:





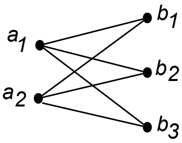
Special simple graphs III: Complete Bipartite Graphs

A **complete bipartite** graph is a simple graph whose vertices may be partitioned into two sets A and B such that:

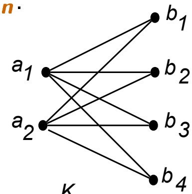
- $\forall a \in A$ and $\forall b \in B$, the edge $\{a, b\}$ belongs to the graph.
- There are no other edges in the graph.

If A has m vertices and B has n vertices, the complete bipartite graph on A and B is denoted by $\boldsymbol{K_{m,n}}$.

Examples:



$K_{2,3}$

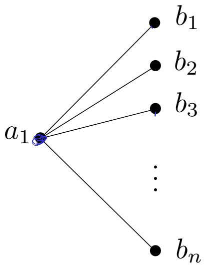


$K_{2,4}$

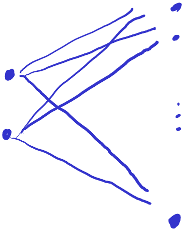
Caution: The name "Complete Bipartite Graph" is misleading. Except for $K_{1,1}$, such graphs are **not complete graphs**.
The adjective 'complete' is qualifies 'bipartite', not 'graph'.

How many edges in $K_{1,n}$?

- *How many edges in the complete bipartite graph $K_{1,n}$?* n



How many edges in $K_{m,n}$?



- *How many edges in* $K_{1,n}$*?*
- *How many edges in* $K_{2,n}$*?* $2n$

$\vdots$

- *How many edges in* $K_{m,n}$*?* mn

Subgraphs

- A **subgraph**, S, of a graph G, is a graph whose vertices are a subset of $V(G)$ and whose edges are a subset of $E(G)$, i.e.

$$V(S) \subseteq V(G)$$
$$E(S) \subseteq E(G)$$

Subgraphs

- A **subgraph**, S, of a graph G, is a graph whose vertices are a subset of $V(G)$ and whose edges are a subset of $E(G)$, i.e.

$$V(S) \subseteq V(G)$$
$$E(S) \subseteq E(G)$$

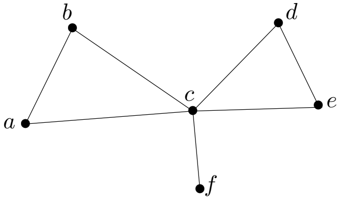
G is a subgraph
of G

- Since S is a graph, if the edge

$$\{a, b\} \in E(S),$$

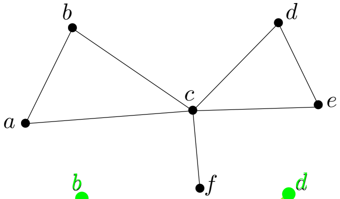
we require its endpoints to be in $V(S)$, i.e. $a, b \in V(S)$.

A subgraph example:

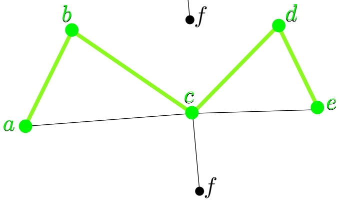
Let G be the graph: 

A subgraph example:

Let G be the graph:

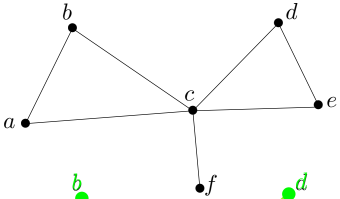


Select some edges and vertices:

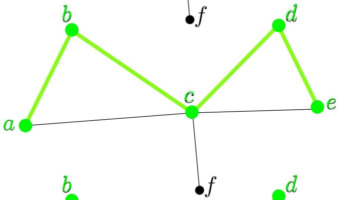


A subgraph example:

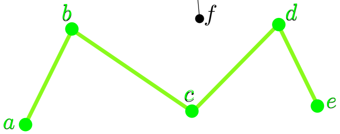
Let G be the graph:



Select some edges and vertices:



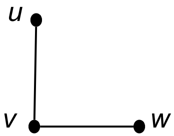
Now S is a subgraph of G:



Another subgraphs example:

Let

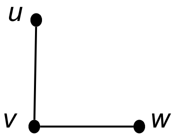
$G =$



Another subgraphs example:

Let

$G =$

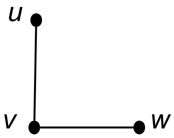


The following five graphs (numbered 1-5) are each subgraphs of G:

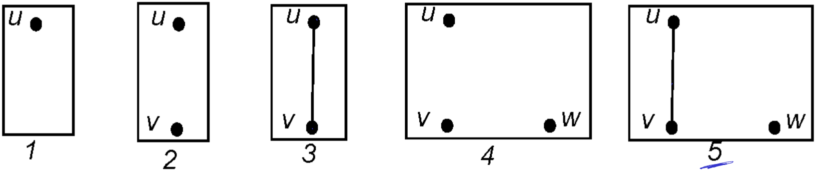
Another subgraphs example:

Let

G =



The following five graphs (numbered 1-5) are each subgraphs of G:

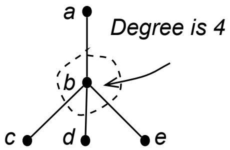


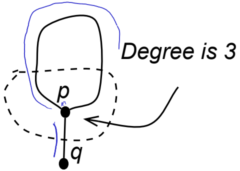
Degree of a vertex

- The **degree** of a vertex is the number of edges incident on it (but with each loop counted twice — once for each ‘end’).

Degree of a vertex

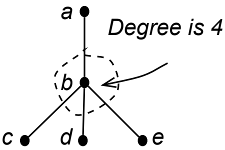
- The **degree** of a vertex is the number of edges incident on it (but with each loop counted twice — once for each 'end').

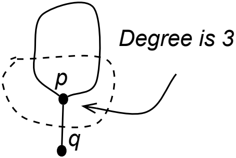




Degree of a vertex

- The **degree** of a vertex is the number of edges incident on it (but with each loop counted twice — once for each ‘end’).

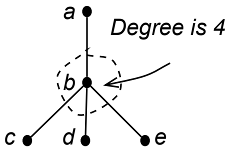


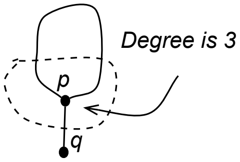


- Let v be a vertex and let k be its degree. We write **deg**$(v) = k$.

Degree of a vertex

- The **degree** of a vertex is the number of edges incident on it (but with each loop counted twice — once for each 'end').

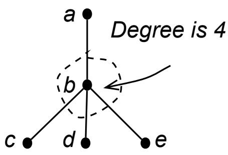


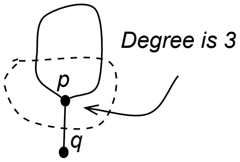


- Let v be a vertex and let k be its degree. We write **deg**$(v)=k$.
- So in the above examples $\deg(b) = 4$ and $\deg(p) = 3$.

Degree of a vertex

- The **degree** of a vertex is the number of edges incident on it (but with each loop counted twice — once for each ‘end’).



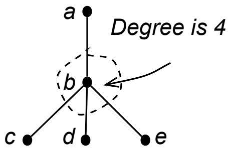


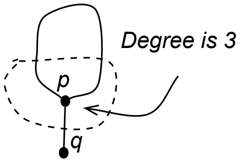
- Let v be a vertex and let k be its degree. We write **deg**$(v)=k$.
- So in the above examples $\deg(b) = 4$ and $\deg(p) = 3$.

Warning: Some authors count a loop as only adding one to a degree, rather than adding two as we do.

Degree of a vertex

- The **degree** of a vertex is the number of edges incident on it (but with each loop counted twice — once for each 'end').





- Let v be a vertex and let k be its degree. We write **deg**$(v)=k$.

- So in the above examples $\deg(b) = 4$ and $\deg(p) = 3$.

Warning: Some authors count a loop as only adding one to a degree, rather than adding two as we do.
We shall always use the 'adding two' version.

Total degree of a graph

The **total degree** of a graph G is the sum of the degrees of all its vertices, $\sum_{v \in V(G)} \deg(v)$.

Total degree of a graph

The **total degree** of a graph G is the sum of the degrees of all its vertices, $\sum_{v \in V(G)} \deg(v)$. *In other words:*

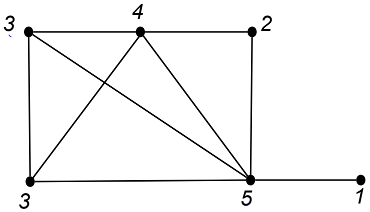
Let $v_1, v_2, \ldots, v_n$ for $n \in \mathbb{N}$, be the vertices of G. Then
the total degree of $G = \deg(v_1) + \deg(v_2) + \cdots + \deg(v_n)$.

Total degree of a graph

The **total degree** of a graph G is the sum of the degrees of all its vertices, $\sum_{v \in V(G)} \deg(v)$. *In other words:*

Let $v_1, v_2, \ldots, v_n$ for $n \in \mathbb{N}$, be the vertices of G. Then
the total degree of $G = \deg(v_1) + \deg(v_2) + \cdots + \deg(v_n)$.

Example: In this graph the degree of each vertex is shown.



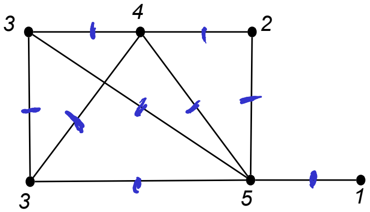
$\deg(G) = 18$

Total degree of a graph

The **total degree** of a graph G is the sum of the degrees of all its vertices, $\sum_{v \in V(G)} \deg(v)$. *In other words:*

Let $v_1, v_2, \ldots, v_n$ for $n \in \mathbb{N}$, be the vertices of G. Then
the total degree of $G = \deg(v_1) + \deg(v_2) + \cdots + \deg(v_n)$.

Example: In this graph the degree of each vertex is shown.



edges = 9

The total degree of the graph is $3 + 4 + 2 + 3 + 5 + 1 = 18$.

The Handshake Theorem

The Handshake Theorem: If G is any graph, then the total degree of G equals twice the number of edges of G.

The Handshake Theorem

The Handshake Theorem: If G is any graph, then the total degree of G equals twice the number of edges of G.

i.e. $$\boxed{\sum_{v \in V(G)} \deg(v) = 2|E(G)|}$$

The Handshake Theorem

The Handshake Theorem: If G is any graph, then the total degree of G equals twice the number of edges of G.

i.e.

$$\boxed{\sum_{v \in V(G)} \deg(v) = 2|E(G)|}$$

Why?

The Handshake Theorem

The Handshake Theorem: If G is any graph, then the total degree of G equals twice the number of edges of G.

i.e. $$\boxed{\sum_{v \in V(G)} \deg(v) = 2|E(G)|}$$

Why?

Because when we count degrees we are counting edges, but we count both ends of each edge, hence we count all the edges twice.

The Handshake Theorem

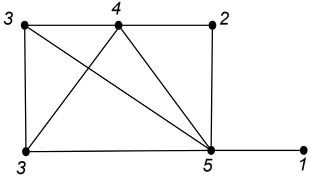
The Handshake Theorem: If G is any graph, then the total degree of G equals twice the number of edges of G.

i.e. $$\boxed{\sum_{v \in V(G)} \deg(v) = 2|E(G)|}$$

Why?

Because when we count degrees we are counting edges, but we count both ends of each edge, hence we count all the edges twice.

Example:



Total degree $= 18$

Number of edges $= 9.$

A Corollary

$\boxed{\text{In any graph there is an even number of vertices of odd degree.}}$

A Corollary

$\boxed{\text{In any graph there is an even number of vertices of odd degree.}}$

How does this corollary follow from the handshake theorem?

A Corollary

$\boxed{\text{In any graph there is an even number of vertices of odd degree.}}$

How does this corollary follow from the handshake theorem?

Example:

- The set $\{1, 1, 2, 3\}$ cannot possibly be the set of degrees of the vertices of some graph.

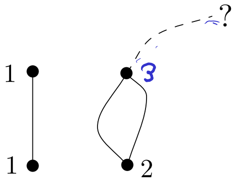
A Corollary

$\boxed{\text{In any graph there is an even number of vertices of odd degree.}}$

How does this corollary follow from the handshake theorem?

Example:

- The set $\{1, 1, 2, 3\}$ cannot possibly be the set of degrees of the vertices of some graph.
- However we try, we always end up with an edge that doesn't have a vertex to connect to:



A useful abbreviation

We often abbreviate

- the graph edge $\{a, b\}$ as ab, and also
- the digraph arc (a, b) as ab.

A useful abbreviation

We often abbreviate

- the graph edge $\{a, b\}$ as ab, and also
- the digraph arc (a, b) as ab.

Note that

- for graphs, ab means *the same* as ba, since $\{a, b\} = \{b, a\}$; but
- for digraphs, ab is *different* from ba since $(a, b) \neq (b, a)$.

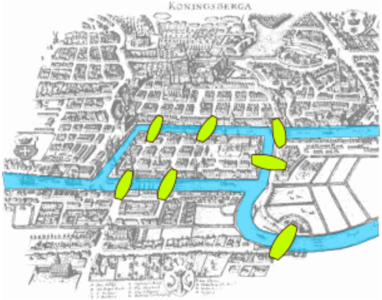
Walks on Graphs

The bridges of Königsberg

- The city of Königsberg in Prussia was set on both sides of the Pregel River, and included two large islands which were connected to each other and the mainland by seven bridges.

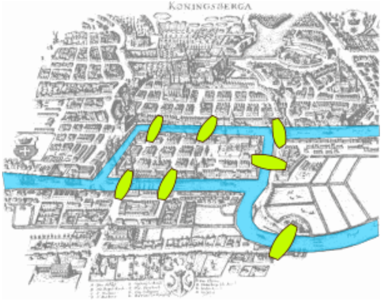
The bridges of Königsberg

- The city of Königsberg in Prussia was set on both sides of the Pregel River, and included two large islands which were connected to each other and the mainland by seven bridges.
- The problem was to find a walk through the city that would cross each bridge once and only once.



The bridges of Königsberg

- The city of Königsberg in Prussia was set on both sides of the Pregel River, and included two large islands which were connected to each other and the mainland by seven bridges.
- The problem was to find a walk through the city that would cross each bridge once and only once.

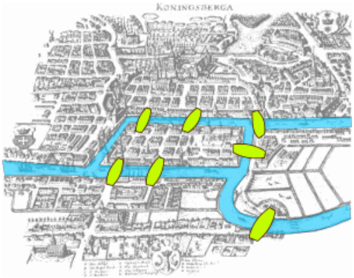


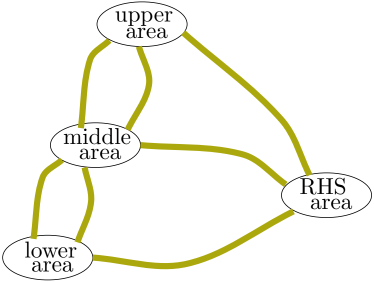
Adapted from:

`http://en.wikipedia.org/wiki/Bridges_of_Konigsberg`

The bridges of Königsberg

Leonard Euler realized that the task can be modeled as a problem in graph theory, which he invented for the purpose.





Adapted from:

`http://en.wikipedia.org/wiki/Bridges_of_Konigsberg`

Walks

- A **walk** in a graph is a sequence of vertices alternating with edges:

$$v_0, e_1, v_1, e_2, v_2, e_3, \ldots, e_n, v_n$$

in which each edge e_k has endpoints v_{k-1} and v_k.

Walks

- A **walk** in a graph is a sequence of vertices alternating with edges:

$$v_0, e_1, v_1, e_2, v_2, e_3, \ldots, e_n, v_n$$

  in which each edge e_k has endpoints v_{k-1} and v_k.

- A walk starting at v_0 and ending at v_n is said to **connect** v_0 to v_n.

Walks

- A **walk** in a graph is a sequence of vertices alternating with edges:

$$v_0, e_1, v_1, e_2, v_2, e_3, \ldots, e_n, v_n$$

  in which each edge e_k has endpoints v_{k-1} and v_k.

- A walk starting at v_0 and ending at v_n is said to **connect** v_0 to v_n.

- The **length** of the walk is the number of edges listed; length n for the walk above.

Walks

- A **walk** in a graph is a sequence of vertices alternating with edges:

$$v_0, e_1, v_1, e_2, v_2, e_3, \ldots, e_n, v_n$$

  in which each edge e_k has endpoints v_{k-1} and v_k.

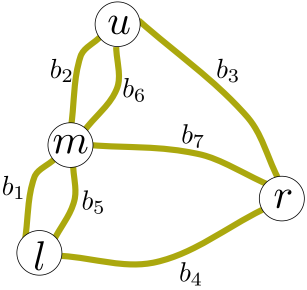
- A walk starting at v_0 and ending at v_n is said to **connect** v_0 to v_n.

- The **length** of the walk is the number of edges listed; length n for the walk above.

- A **trivial walk**, say v_0, contains no edges; hence has length 0.

Walks on the Königsberg Graph

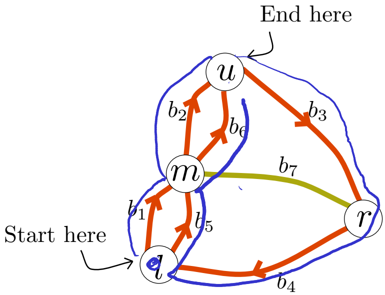
The question becomes, 'Is there a walk on the Königsberg Graph which traverses each edge exactly once?'



Walks on the Königsberg Graph

Try starting in the lower part of town, going via bridge b_1 to the middle island, then via bridge b_2 to the upper part, then via bridge b_3 to the right-most part; continuing as in the listed walk:

$$l b_1 m b_2 u b_3 r b_4 l b_5 m b_6 u$$

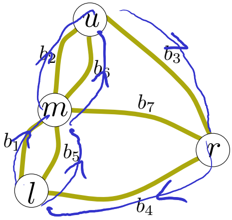


Walks on the Königsberg Graph

- That attempt didn't work, because there is no unused edge to exit u from on the second visit.

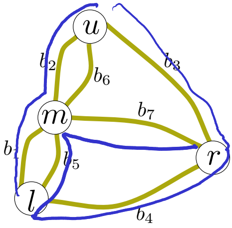
Walks on the Königsberg Graph

- That attempt didn't work, because there is no unused edge to exit u from on the second visit.
- For brevity we'll leave out vertices and just list edges. Try
  - $b_1 b_2 b_3 b_4 b_5 b_6$ ... stuck at u



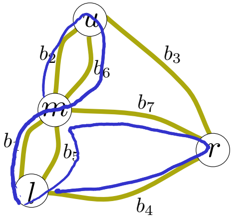
Walks on the Königsberg Graph

- That attempt didn't work, because there is no unused edge to exit u from on the second visit.
- For brevity we'll leave out vertices and just list edges. Try
  - $b_1 b_2 b_3 b_4 b_5 b_6$ ... stuck at u
  - $b_1 b_2 b_3 b_4 b_5 b_7$ ... stuck at r



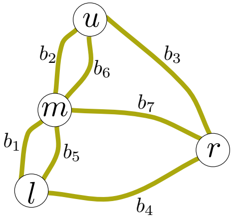
Walks on the Königsberg Graph

- That attempt didn't work, because there is no unused edge to exit u from on the second visit.
- For brevity we'll leave out vertices and just list edges. Try
  - $b_1 b_2 b_3 b_4 b_5 b_6$ ... stuck at u
  - $b_1 b_2 b_3 b_4 b_5 b_7$ ... stuck at r
  - $b_2 b_6 b_1 b_5 b_7 b_4$ ... stuck at l



Walks on the Königsberg Graph

- That attempt didn't work, because there is no unused edge to exit u from on the second visit.
- For brevity we'll leave out vertices and just list edges. Try
  - $b_1 b_2 b_3 b_4 b_5 b_6$ ... stuck at u
  - $b_1 b_2 b_3 b_4 b_5 b_7$ ... stuck at r
  - $b_2 b_6 b_1 b_5 b_7 b_4$ ... stuck at l
  - $b_1 b_2 b_6 b_5 b_4 b_7$ ... stuck at m



Impossibility of Königsberg Bridge Walk

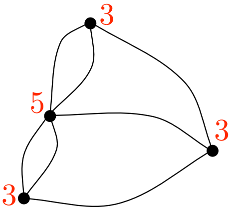
- The kind of walk needed to meet the criteria for the Königsberg Bridge problem - a walk which traverses all edges and repeats none - came to be called an **Euler Path**.

Impossibility of Königsberg Bridge Walk

- The kind of walk needed to meet the criteria for the Königsberg Bridge problem - a walk which traverses all edges and repeats none - came to be called an **Euler Path**.
- Euler showed that no such walk exists on the Königsberg Graph.

Impossibility of Königsberg Bridge Walk

- The kind of walk needed to meet the criteria for the Königsberg Bridge problem - a walk which traverses all edges and repeats none - came to be called an **Euler Path**.
- Euler showed that no such walk exists on the Königsberg Graph.
- *How? Think about the degrees of the vertices:*



Impossibility of Königsberg Bridge Walk

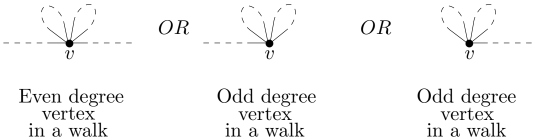
- *There are too many vertices of odd degree in the Königsberg Graph for an Euler path to be possible. Can you prove it?*

Impossibility of Königsberg Bridge Walk

- *There are too many vertices of odd degree in the Königsberg Graph for an Euler path to be possible. Can you prove it?*
- *Hint: can an odd degree vertex occur in the middle of a walk?*

Impossibility of Königsberg Bridge Walk

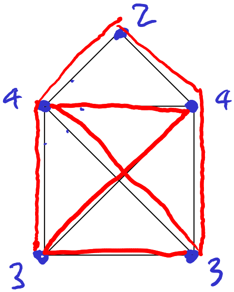
- *There are too many vertices of odd degree in the Königsberg Graph for an Euler path to be possible. Can you prove it?*
- *Hint: can an odd degree vertex occur in the middle of a walk?*



A related puzzle

You may have come across the following puzzle:

Can you draw the following 'house' diagram without taking your pen off the paper or overwriting edges?

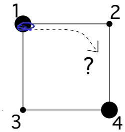


Can you?

Yes, start and end at an odd degree vertex.

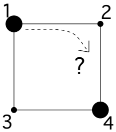
Walks, paths and circuits

Counting Walks with Adjacency Matrices

The graph 

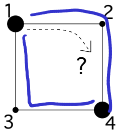
has adjacency matrix $M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \!\!\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}.$

Counting Walks with Adjacency Matrices

The graph  has adjacency matrix $M = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & \multicolumn{4}{c}{\multirow{4}{*}{$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$}} \\ 2 \\ 3 \\ 4 \end{array}$.

There are 0 ways to walk from vertex 1 to vertex 4 in a single step.

Counting Walks with Adjacency Matrices

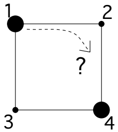
The graph 

has adjacency matrix $M = \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}.$

There are 0 ways to walk from vertex 1 to vertex 4 in a single step.

There are 2 ways to walk from vertex 1 to vertex 4 in a two steps:

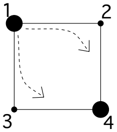
Counting Walks with Adjacency Matrices

The graph 

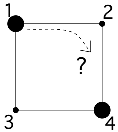
has adjacency matrix $M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}.$

There are 0 ways to walk from vertex 1 to vertex 4 in a single step.

There are 2 ways to walk from vertex 1 to vertex 4 in a two steps:

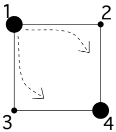


Counting Walks with Adjacency Matrices

The graph  has adjacency matrix
$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}.$$

There are 0 ways to walk from vertex 1 to vertex 4 in a single step.

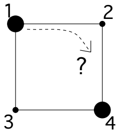
There are 2 ways to walk from vertex 1 to vertex 4 in a two steps:



$$M^2 = \begin{array}{c} \begin{array}{cccc} 1\text{-}1 & 1\text{-}2 & 1\text{-}3 & 1\text{-}4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \begin{array}{c} \begin{array}{cccc} & & & 1\text{-}4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \begin{array}{c} \\ \\ 2\text{-}4 \\ 3\text{-}4 \\ 4\text{-}4 \end{array}$$

$$= \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 2 & 0 & 0 & 2 \\ 0 & 2 & 2 & 0 \\ 0 & 2 & 2 & 0 \\ 2 & 0 & 0 & 2 \end{pmatrix} \end{array}$$

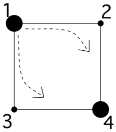
Counting Walks with Adjacency Matrices

The graph 

has adjacency matrix $M = \begin{array}{c@{}c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}.$

There are 0 ways to walk from vertex 1 to vertex 4 in a single step.

There are 2 ways to walk from vertex 1 to vertex 4 in a two steps:



$$M^2 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

$$= \begin{array}{c@{}c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 2 & 0 & 0 & 2 \\ 0 & 2 & 2 & 0 \\ 0 & 2 & 2 & 0 \\ 2 & 0 & 0 & 2 \end{pmatrix} \end{array}$$

For any graph, the number of ways to walk from vertex i to vertex j in t steps is given in terms of its adjacency matrix M by the $(i, j)^{\text{th}}$ entry of M^t.

Some special kinds of walks

Some properties that a walk on a graph may, or may not, possess are:

Some special kinds of walks

Some properties that a walk on a graph may, or may not, possess are:

- being ‘closed’;

Some special kinds of walks

Some properties that a walk on a graph may, or may not, possess are:

- being ‘closed’;
- having no repeated edges;

Some special kinds of walks

Some properties that a walk on a graph may, or may not, possess are:

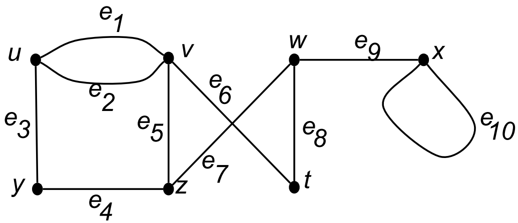
- being ‘closed’;
- having no repeated edges;
- having no repeated vertices.

Some special kinds of walks

Some properties that a walk on a graph may, or may not, possess are:

- being ‘closed’;
- having no repeated edges;
- having no repeated vertices.

We will now look at each of these potential properties in turn, with examples using the graph G below.



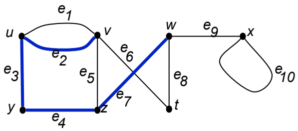
Closed walks

A walk $v_0, e_1, v_1, e_2, \ldots, e_n, v_n$ is called a **closed** when $v_0 = v_n$.

Closed walks

A walk $v_0, e_1, v_1, e_2, \ldots, e_n, v_n$ is called a **closed** when $v_0 = v_n$.

Examples:



The walk

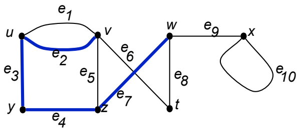
$$v\ \mathbf{e_2}\ u\ \mathbf{e_3}\ y\ \mathbf{e_4}\ z\ \mathbf{e_7}\ w$$

has length 4 and is **not** closed because $v = v_0 \neq v_n = v_4 = w$.

Closed walks

A walk $v_0, e_1, v_1, e_2, \ldots, e_n, v_n$ is called a **closed** when $v_0 = v_n$.

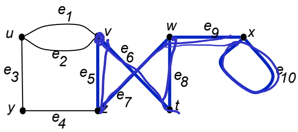
Examples:



The walk

$$v\ \mathbf{e_2}\ u\ \mathbf{e_3}\ y\ \mathbf{e_4}\ z\ \mathbf{e_7}\ w$$

has length 4 and is **not** closed because $v = v_0 \neq v_n = v_4 = w$.



The walk

$$v\ \mathbf{e_6}\ t\ \mathbf{e_8}\ w\ \mathbf{e_9}\ x\ \mathbf{e_{10}}\ x\ \mathbf{e_9}\ w\ \mathbf{e_7}\ z\ \mathbf{e_5}\ v$$

has length 7 and **is** closed because $v = v_0 = v_n = v_7 = v$.

Paths

A **path** is a walk that does not repeat any edge.

Paths

A **path** is a walk that does not repeat any edge.

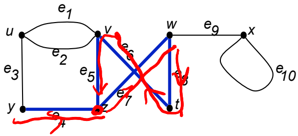
A **simple path** is a path which does not repeat any vertex — except the first and last if the path is closed.

Paths

A **path** is a walk that does not repeat any edge.

A **simple path** is a path which does not repeat any vertex — except the first and last if the path is closed.

Examples:



The walk

$$y\ \mathbf{e_4}\ z\ \mathbf{e_7}\ w\ \mathbf{e_8}\ t\ \mathbf{e_6}\ v\ \mathbf{e_5}\ z$$

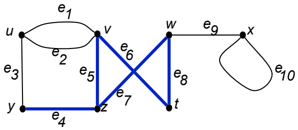
has length 5 and is a **path** because the five edges are all different, but is **not simple** because $z = v_1 = v_5$

Paths

A **path** is a walk that does not repeat any edge.

A **simple path** is a path which does not repeat any vertex — except the first and last if the path is closed.

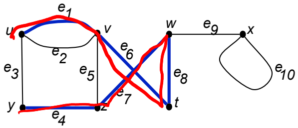
Examples:



The walk

$$y\ \mathbf{e_4}\ z\ \mathbf{e_7}\ w\ \mathbf{e_8}\ t\ \mathbf{e_6}\ v\ \mathbf{e_5}\ z$$

has length 5 and is a **path** because the five edges are all different, but is **not simple** because $z = v_1 = v_5$



The walk

$$y\ \mathbf{e_4}\ z\ \mathbf{e_7}\ w\ \mathbf{e_8}\ t\ \mathbf{e_6}\ v\ \mathbf{e_1}\ u$$

has length 5 and is a **simple path** because the six vertices are all different.

Circuits

A **circuit** is a closed path.

Circuits

A **circuit** is a closed path.

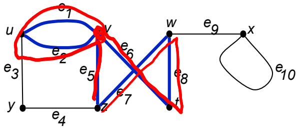
A **simple circuit** is a simple closed path.

Circuits

A **circuit** is a closed path.

A **simple circuit** is a simple closed path.

Examples:



The walk

$$z\ \mathbf{e_7}\ w\ \mathbf{e_8}\ t\ \mathbf{e_6}\ v\ \mathbf{e_1}\ u\ \mathbf{e_2}\ v\ \mathbf{e_5}\ z$$

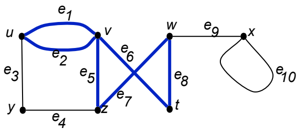
has length 6 and is a **circuit** as it is closed with all different edges, but is **not simple** because $v = v_3 = v_5$.

Circuits

A **circuit** is a closed path.

A **simple circuit** is a simple closed path.

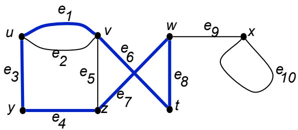
Examples:



The walk

$$z\ \mathbf{e_7}\ w\ \mathbf{e_8}\ t\ \mathbf{e_6}\ v\ \mathbf{e_1}\ u\ \mathbf{e_2}\ v\ \mathbf{e_5}\ z$$

has length 6 and is a **circuit** as it is closed with all different edges, but is **not simple** because $v = v_3 = v_5$.



The walk

$$z\ \mathbf{e_7}\ w\ \mathbf{e_8}\ t\ \mathbf{e_6}\ v\ \mathbf{e_1}\ u\ \mathbf{e_3}\ y\ \mathbf{e_4}\ z$$

has length 6 and is a **simple circuit** as it is closed without repeated vertices except the first and last $z = v_0 = v_6$.

Walks on digraphs

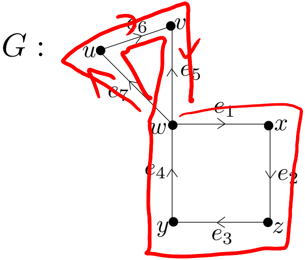
- A walk in a **digraph** is sometimes called a **directed walk**.

Walks on digraphs

- A walk in a **digraph** is sometimes called a **directed walk**.
- In a directed walk, the sequence of edges alone always uniquely determines the path.

Walks on digraphs

- A walk in a **digraph** is sometimes called a **directed walk**.
- In a directed walk, the sequence of edges alone always uniquely determines the path. Examples:



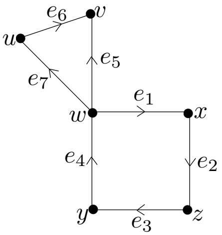
- $e_1e_2e_3e_4e_7e_6$ is a directed walk in G
- $e_1e_2e_3e_4$ is a closed directed walk in G
- $\cancel{e_7e_6e_5}$ is NOT a directed walk in G

Doesn't follow the direction of the edges.

Walks on digraphs

- A walk in a **digraph** is sometimes called a **directed walk**.
- In a directed walk, the sequence of edges alone always uniquely determines the path. Examples:

G :



- $e_1e_2e_3e_4e_7e_6$ is a directed walk in G
- $e_1e_2e_3e_4$ is a closed directed walk in G
- $\cancel{e_7e_6}e_5$ is NOT a directed walk in G

- Note that for a simple graph or digraph (with no loops and no parallel edges), **any** walk is uniquely determined by its sequence of vertices.

Connected Graphs

- A graph is **connected** if every pair of vertices can be connected by a walk (and therefore by a path).

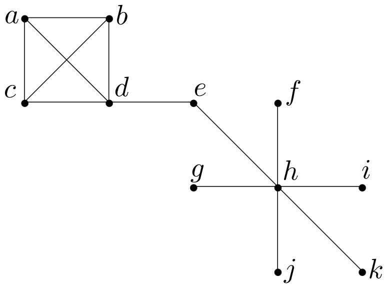
Connected Graphs

- A graph is **connected** if every pair of vertices can be connected by a walk (and therefore by a path).
- A **component** of a graph is a maximal connected subgraph.

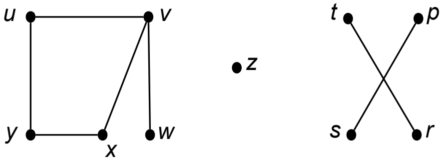
Connected Graphs

- A graph is **connected** if every pair of vertices can be connected by a walk (and therefore by a path).
- A **component** of a graph is a maximal connected subgraph.

Here is an example of a connected graph:

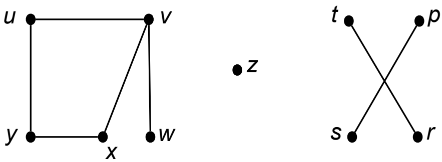


Example



The graph above is **not connected** and has **4 components**:

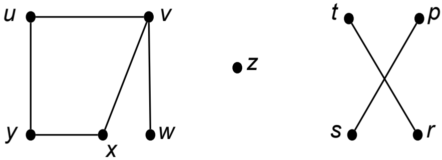
Example



The graph above is **not connected** and has **4 components**:

1. the subgraph with vertex set $\{u, v, w, x, y\}$ and edge set $\{\{u, v\}, \{u, y\}, \{v, w\}, \{v, x\}, \{x, y\}\}$,

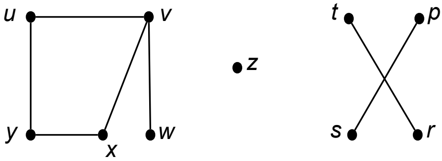
Example



The graph above is **not connected** and has **4 components**:

1. the subgraph with vertex set $\{u, v, w, x, y\}$ and edge set $\{\{u, v\}, \{u, y\}, \{v, w\}, \{v, x\}, \{x, y\}\}$,
2. the subgraph with vertex z and no edges,

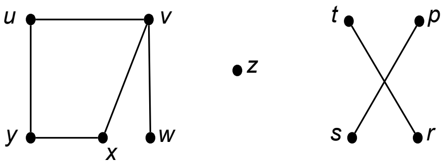
Example



The graph above is **not connected** and has **4 components**:

1. the subgraph with vertex set $\{u, v, w, x, y\}$ and edge set $\{\{u, v\}, \{u, y\}, \{v, w\}, \{v, x\}, \{x, y\}\}$,
2. the subgraph with vertex z and no edges,
3. the subgraph with two vertices t, r and one edge $\{t, r\}$,

Example



The graph above is **not connected** and has **4 components**:

1. the subgraph with vertex set $\{u, v, w, x, y\}$ and edge set $\{\{u, v\}, \{u, y\}, \{v, w\}, \{v, x\}, \{x, y\}\}$,
2. the subgraph with vertex z and no edges,
3. the subgraph with two vertices t, r and one edge $\{t, r\}$,
4. the subgraph with two vertices p, s and one edge $\{p, s\}$.

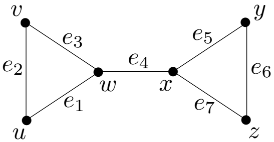
Bridges and Cut Vertices

- A **bridge** in a connected graph is an edge which on erasure disconnects the graph.

Bridges and Cut Vertices

- A **bridge** in a connected graph is an edge which on erasure disconnects the graph. Examples:

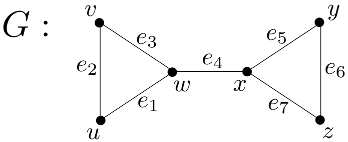
G :



  - e_4 is a bridge in G
  - e_3 is NOT a bridge in G

Bridges and Cut Vertices

- A **bridge** in a connected graph is an edge which on erasure disconnects the graph. Examples:

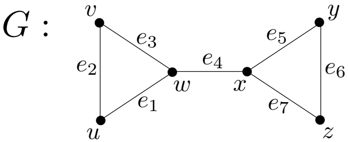


  - e_4 is a bridge in G
  - e_3 is NOT a bridge in G

- A **cut vertex** in a connected graph is a vertex which on erasure disconnects the graph.

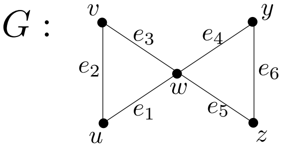
Bridges and Cut Vertices

- A **bridge** in a connected graph is an edge which on erasure disconnects the graph. Examples:



  - e_4 is a bridge in G
  - e_3 is NOT a bridge in G

- A **cut vertex** in a connected graph is a vertex which on erasure disconnects the graph. Examples:



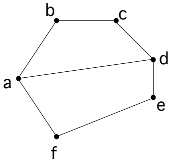
  - w is a cut vertex in G
  - v is NOT a cut vertex in G

Euler Paths and Circuits

- An **Euler path** for a graph is a path passing through every edge (hence every vertex). (Recall the Königsberg Bridge problem.)

Euler Paths and Circuits

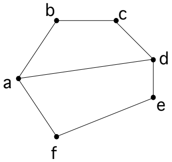
- An **Euler path** for a graph is a path passing through every edge (hence every vertex). (Recall the Königsberg Bridge problem.) Example:



This graph has an Euler path: $\boxed{abcdafed}$.

Euler Paths and Circuits

- An **Euler path** for a graph is a path passing through every edge (hence every vertex). (Recall the Königsberg Bridge problem.) Example:

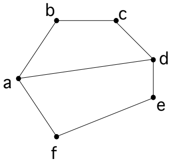


This graph has an Euler path: $\boxed{abcdafed}$.

- An **Euler circuit** for a graph is a circuit passing through every edge (hence every vertex).

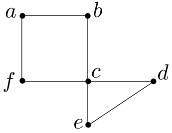
Euler Paths and Circuits

- An **Euler path** for a graph is a path passing through every edge (hence every vertex). (Recall the Königsberg Bridge problem.) Example:



This graph has an Euler path: $\boxed{abcdafed}$.

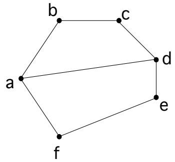
- An **Euler circuit** for a graph is a circuit passing through every edge (hence every vertex). Example:



This graph has an Euler circuit: $\boxed{abcedcfa.}$

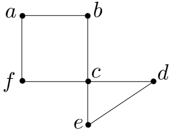
Euler Paths and Circuits

- An **Euler path** for a graph is a path passing through every edge (hence every vertex). (Recall the Königsberg Bridge problem.) Example:



This graph has an Euler path: $\boxed{abcdafed}$.

- An **Euler circuit** for a graph is a circuit passing through every edge (hence every vertex). Example:



This graph has an Euler circuit: $\boxed{abcedcfa.}$

Note: By convention, an Euler path must be open, *i.e* not a circuit.

When do Euler Paths and Circuits exist?

- **Theorem:** A connected graph has an Euler circuit if and only if each of its vertices has even degree.

When do Euler Paths and Circuits exist?

- **Theorem:** A connected graph has an Euler circuit if and only if each of its vertices has even degree.

  In this case we will give an algorithm for finding an Euler circuit.

When do Euler Paths and Circuits exist?

- **Theorem:** A connected graph has an Euler circuit if and only if each of its vertices has even degree.

  In this case we will give an algorithm for finding an Euler circuit.

- **Corollary:** A connected graph has an Euler path if and only if it has exactly two vertices of odd degree.

When do Euler Paths and Circuits exist?

- **Theorem:** A connected graph has an Euler circuit if and only if each of its vertices has even degree.

  In this case we will give an algorithm for finding an Euler circuit.

- **Corollary:** A connected graph has an Euler path if and only if it has exactly two vertices of odd degree.

  The algorithm easily adapts to this case.

Issues regarding finding an Euler Circuit in a graph G

- By the theorem, G must be connected and have all its vertices of even degree. Given this, an Euler circuit must exist.

Issues regarding finding an Euler Circuit in a graph G

- By the theorem, G must be connected and have all its vertices of even degree. Given this, an Euler circuit must exist.
- On a circuit, you can start anywhere and return after completing the circuit. So start the search at any vertex in G.

Issues regarding finding an Euler Circuit in a graph G

- By the theorem, G must be connected and have all its vertices of even degree. Given this, an Euler circuit must exist.
- On a circuit, you can start anywhere and return after completing the circuit. So start the search at any vertex in G.
- As you walk through G mark each edge you use; it cannot be used again. Imagine the edge has been erased, so that graph is 'reduced'.

Issues regarding finding an Euler Circuit in a graph G

- By the theorem, G must be connected and have all its vertices of even degree. Given this, an Euler circuit must exist.
- On a circuit, you can start anywhere and return after completing the circuit. So start the search at any vertex in G.
- As you walk through G mark each edge you use; it cannot be used again. Imagine the edge has been erased, so that graph is 'reduced'.
- After 'erasing' an edge, the start vertex and the 'current' vertex have odd degree in the reduced graph, while all other vertices remain with even degree. You then seek an Euler **path** (in the reduced graph) from the current vertex to the start vertex. By the corollary, such a path exists provided the reduced graph is connected.

Issues regarding finding an Euler Circuit in a graph G

- By the theorem, G must be connected and have all its vertices of even degree. Given this, an Euler circuit must exist.
- On a circuit, you can start anywhere and return after completing the circuit. So start the search at any vertex in G.
- As you walk through G mark each edge you use; it cannot be used again. Imagine the edge has been erased, so that graph is ‘reduced’.
- After ‘erasing’ an edge, the start vertex and the ’current’ vertex have odd degree in the reduced graph, while all other vertices remain with even degree. You then seek an Euler **path** (in the reduced graph) from the current vertex to the start vertex. By the corollary, such a path exists provided the reduced graph is connected.
- So choose each edge so that the reduced graph is still connected.

Issues regarding finding an Euler Circuit in a graph G

- By the theorem, G must be connected and have all its vertices of even degree. Given this, an Euler circuit must exist.
- On a circuit, you can start anywhere and return after completing the circuit. So start the search at any vertex in G.
- As you walk through G mark each edge you use; it cannot be used again. Imagine the edge has been erased, so that graph is 'reduced'.
- After 'erasing' an edge, the start vertex and the 'current' vertex have odd degree in the reduced graph, while all other vertices remain with even degree. You then seek an Euler **path** (in the reduced graph) from the current vertex to the start vertex. By the corollary, such a path exists provided the reduced graph is connected.
- So choose each edge so that the reduced graph is still connected.
- Always leave an edge to return to the start vertex as the last step.

Fleury's Algorithm for finding Euler Circuits

Let G be a connected graph with all vertices of even degree.

Fleury's Algorithm for finding Euler Circuits

Let G be a connected graph with all vertices of even degree.

The Algorithm

1. Pick any vertex of G as a starting point.

Fleury's Algorithm for finding Euler Circuits

Let G be a connected graph with all vertices of even degree.

The Algorithm

1. Pick any vertex of G as a starting point.
2. From that vertex **choose any edge to traverse that does not cross a bridge of the current reduced graph, unless there is no other choice**. ("No choice" will only happen when the vertex has degree 1 in the current reduced graph. The new reduced graph will then omit that vertex and so remain connected.)

Fleury's Algorithm for finding Euler Circuits

Let G be a connected graph with all vertices of even degree.

The Algorithm

1. Pick any vertex of G as a starting point.
2. From that vertex **choose any edge to traverse that does not cross a bridge of the current reduced graph, unless there is no other choice**. ("No choice" will only happen when the vertex has degree 1 in the current reduced graph. The new reduced graph will then omit that vertex and so remain connected.)
3. Mark the edge (*e.g.* darken it) as a reminder not to traverse it again. Treat the edge as erased, so reducing the graph.

Fleury's Algorithm for finding Euler Circuits

Let G be a connected graph with all vertices of even degree.

The Algorithm

1. Pick any vertex of G as a starting point.
2. From that vertex **choose any edge to traverse that does not cross a bridge of the current reduced graph, unless there is no other choice**. ("No choice" will only happen when the vertex has degree 1 in the current reduced graph. The new reduced graph will then omit that vertex and so remain connected.)
3. Mark the edge (*e.g.* darken it) as a reminder not to traverse it again. Treat the edge as erased, so reducing the graph.
4. Travel that edge, coming to the next vertex.

Fleury's Algorithm for finding Euler Circuits

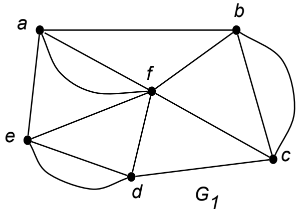
Let G be a connected graph with all vertices of even degree.

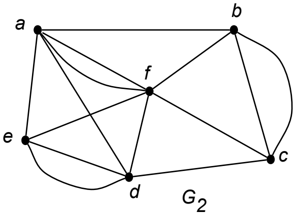
The Algorithm

1. Pick any vertex of G as a starting point.
2. From that vertex **choose any edge to traverse that does not cross a bridge of the current reduced graph, unless there is no other choice**. ("No choice" will only happen when the vertex has degree 1 in the current reduced graph. The new reduced graph will then omit that vertex and so remain connected.)
3. Mark the edge (*e.g.* darken it) as a reminder not to traverse it again. Treat the edge as erased, so reducing the graph.
4. Travel that edge, coming to the next vertex.
5. Repeat steps 2 - 4 until all edges have been traversed, and you are back to the starting vertex.

Candidate Graphs for Fleury's Algorithm

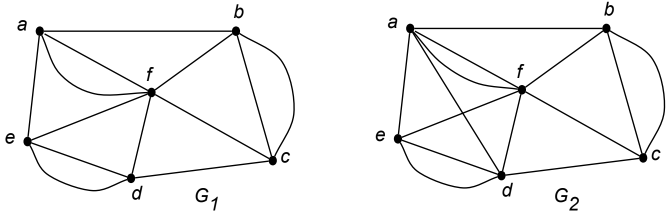
The graph G_1 below satisfies the criterion that all vertices have even degree, so it contains an Euler circuit and Fleury's algorithm can be used to find that circuit.





Candidate Graphs for Fleury's Algorithm

The graph G_1 below satisfies the criterion that all vertices have even degree, so it contains an Euler circuit and Fleury's algorithm can be used to find that circuit.

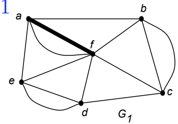


The graph G_2 has two vertices of odd degree. Fleury's algorithm can be modified to find an Euler path in this graph. The only modification needed is that the first vertex must be one of the vertices of odd degree.

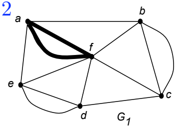
Fleury's Algorithm example in pictures

Start at f (just because we feel like it!)

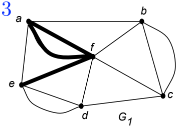
1



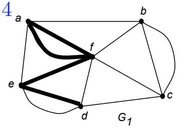
2



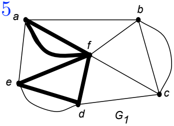
3



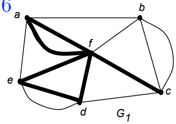
4



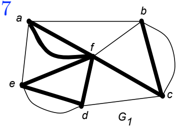
5



6



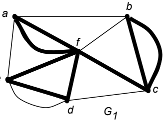
7

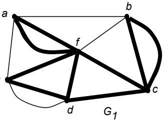


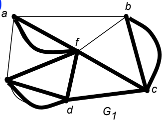
Notice that from here
we MAY NOT step to f
but either a or c is allowed

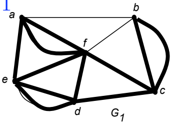
Fleury's Algorithm example in pictures

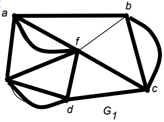
At step 9 we're forced to go to d; and then all steps are forced.

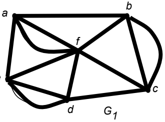
8


9


10


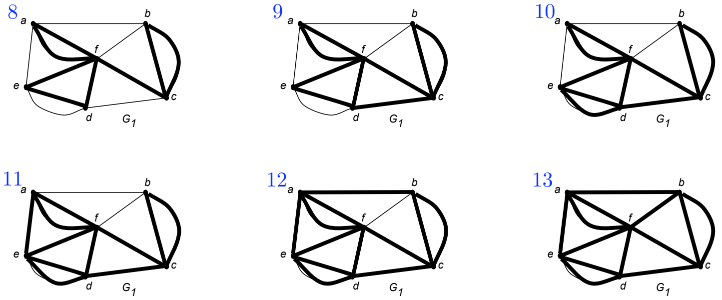
11


12


13


Fleury's Algorithm example in pictures

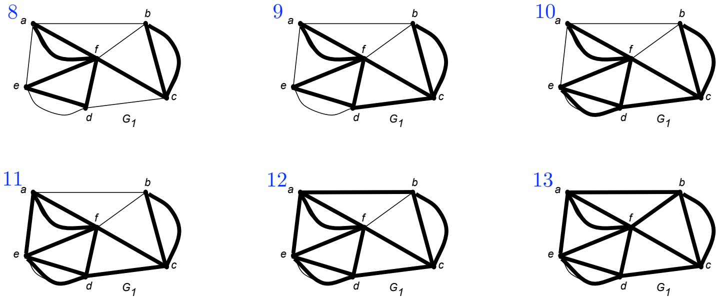
At step 9 we're forced to go to d; and then all steps are forced.



The final path is $fafedfcbcdeabf$.

Fleury's Algorithm example in pictures

At step 9 we're forced to go to d; and then all steps are forced.

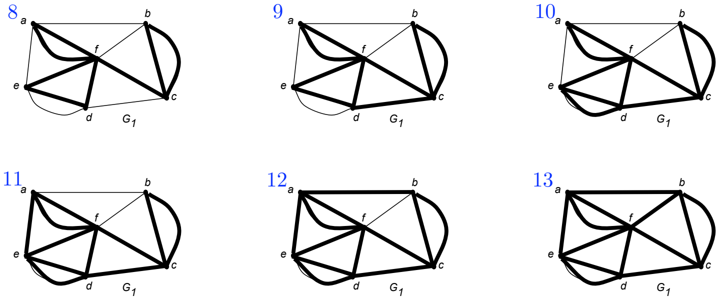


The final path is $fafedfcbcdeabf$.

The path is now closed, providing the Euler circuit we sought.

Fleury's Algorithm example in pictures

At step 9 we're forced to go to d; and then all steps are forced.



The final path is $\qquad fafedfcbcdeabf\,.$

The path is now closed, providing the Euler circuit we sought.

(As mentioned earlier, we do not call this path an Euler *path*, because, by convention, Euler paths ar not closed.)

Fleury's Algorithm example discussion

- Note that if we had started at some other vertex, or made different choices along the way, Fleury's algorithm would have given a different Euler circuit.

Fleury's Algorithm example discussion

- Note that if we had started at some other vertex, or made different choices along the way, Fleury's algorithm would have given a different Euler circuit.
- In any implementation, we never have to back-track, so the algorithm is quite fast; as are some other algorithms to solve this problem.

Fleury's Algorithm example discussion

- Note that if we had started at some other vertex, or made different choices along the way, Fleury's algorithm would have given a different Euler circuit.
- In any implementation, we never have to back-track, so the algorithm is quite fast; as are some other algorithms to solve this problem.
- By contrast, the following problem – that of finding a *Hamilton* path or circuit – has no known 'fast' algorithm.

Hamilton Paths and Circuits

- A **Hamilton path** for a graph is a *simple* path which passes through every vertex.

Hamilton Paths and Circuits

- A **Hamilton path** for a graph is a *simple* path which passes through every vertex.
- A **Hamilton circuit** for a graph is a *simple* circuit which passes through every vertex.

Hamilton Paths and Circuits

- A **Hamilton path** for a graph is a *simple* path which passes through every vertex.
- A **Hamilton circuit** for a graph is a *simple* circuit which passes through every vertex.
- Note that a Hamilton path (respectively circuit)
  - does not have to use every edge,

Hamilton Paths and Circuits

- A **Hamilton path** for a graph is a *simple* path which passes through every vertex.

- A **Hamilton circuit** for a graph is a *simple* circuit which passes through every vertex.

- Note that a Hamilton path (respectively circuit)
  - does not have to use every edge,
  - may use an edge at most once by the definition of path (respectively circuit),

Hamilton Paths and Circuits

- A **Hamilton path** for a graph is a *simple* path which passes through every vertex.
- A **Hamilton circuit** for a graph is a *simple* circuit which passes through every vertex.
- Note that a Hamilton path (respectively circuit)
  - does not have to use every edge,
  - may use an edge at most once by the definition of path (respectively circuit),
  - must use each vertex exactly once by the definition of simple.

Hamilton Paths and Circuits

- A **Hamilton path** for a graph is a *simple* path which passes through every vertex.
- A **Hamilton circuit** for a graph is a *simple* circuit which passes through every vertex.
- Note that a Hamilton path (respectively circuit)
  - does not have to use every edge,
  - may use an edge at most once by the definition of path (respectively circuit),
  - must use each vertex exactly once by the definition of simple.
- A graph is called **Hamiltonian** if and only if it possesses a Hamilton circuit.

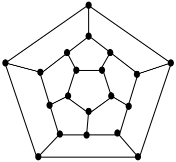
Hamilton Paths and Circuits

- A **Hamilton path** for a graph is a *simple* path which passes through every vertex.
- A **Hamilton circuit** for a graph is a *simple* circuit which passes through every vertex.
- Note that a Hamilton path (respectively circuit)
  - does not have to use every edge,
  - may use an edge at most once by the definition of path (respectively circuit),
  - must use each vertex exactly once by the definition of simple.
- A graph is called **Hamiltonian** if and only if it possesses a Hamilton circuit.

Note: By convention, a Hamilton path must be open, *i.e* not a circuit.

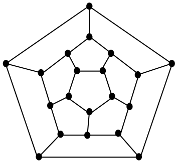
Graphs and Hamilton Paths / Circuits

- This graph has a Hamilton Circuit. *Can you find one?*

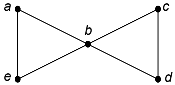


Graphs and Hamilton Paths / Circuits

- This graph has a Hamilton Circuit. *Can you find one?*



- This graph has no Hamilton circuit. *Why not?*



Types of Walks on Graphs – Summary

For a walk $v_0, e_1, v_1, e_2, \ldots, e_n, v_n$ on a graph G:

Properties					
Name:	**closed**	—	**Euler**	**simple**	**Hamilton**
Description:	—	no repeated edges	uses all edges	no repeated vertices	uses all vertices
Requirement:	$v_0 = v_n$	$i \neq j \implies e_i \neq e_j$	$\forall e \in E(G)$ $\exists i\ e_i = e$	$i \neq j \implies v_i \neq v_j$ ($v_0 = v_n$ OK)	$\forall v \in V(G)$ $\exists i\ v_i = v$
path		✓			
simple path		✓		✓	
Euler path		✓	✓		
Hamilton path		✓		✓	✓
closed walk	✓				
circuit	✓	✓			
simple circuit	✓	✓		✓	
Euler circuit	✓	✓	✓		
Hamilton circuit	✓	✓		✓	✓

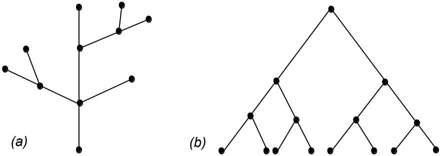
Trees

Trees

- A **tree** is a connected graph with no circuits other that the trivial ones.

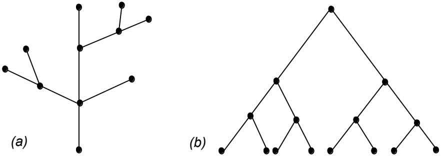
Trees

- A **tree** is a connected graph with no circuits other that the trivial ones. Examples:

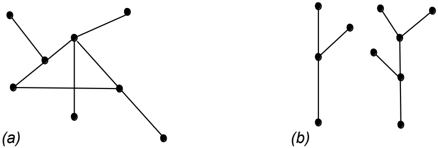


Trees

- A **tree** is a connected graph with no circuits other that the trivial ones. Examples:

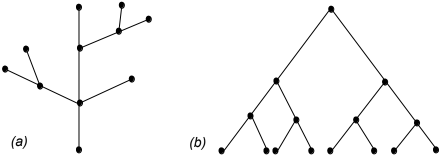


- Examples of **non-trees**

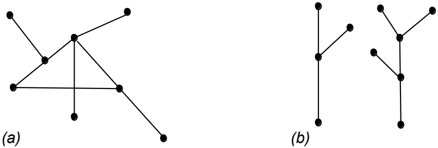


Trees

- A **tree** is a connected graph with no circuits other that the trivial ones. Examples:



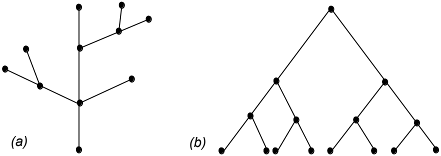
- Examples of **non-trees**



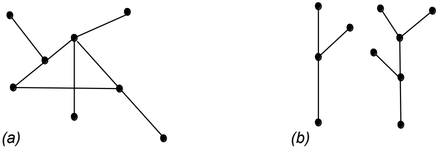
  - Example (a) contains a circuit, so is not a tree.

Trees

- A **tree** is a connected graph with no circuits other that the trivial ones. Examples:



- Examples of **non-trees**



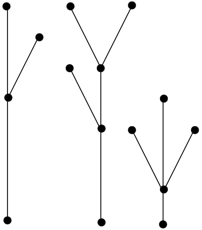
  - Example (a) contains a circuit, so is not a tree.
  - Example (b) is not a tree. *Why not?*

Forests

- A **forest** is a disjoint collection of trees.

Forests

- A **forest** is a disjoint collection of trees. Example:



Trees as Models

- Trees are used as models in such diverse areas as computer science, chemistry, geology, botany, psychology, management, linguistics and many others. (See Epp,10.5[4ed], 10.4[5ed]).

Trees as Models

- Trees are used as models in such diverse areas as computer science, chemistry, geology, botany, psychology, management, linguistics and many others. (See Epp,10.5[4ed], 10.4[5ed]).
- On your computer, the folder and file directory has a tree structure.

Trees as Models

- Trees are used as models in such diverse areas as computer science, chemistry, geology, botany, psychology, management, linguistics and many others. (See Epp,10.5[4ed], 10.4[5ed]).
- On your computer, the folder and file directory has a tree structure.
- Trees are used in linguistics to describe grammatical relationships.

Trees as Models

- Trees are used as models in such diverse areas as computer science, chemistry, geology, botany, psychology, management, linguistics and many others. (See Epp,10.5[4ed], 10.4[5ed]).
- On your computer, the folder and file directory has a tree structure.
- Trees are used in linguistics to describe grammatical relationships.
- In botany and zoology, taxonomy of species is entirely based on a tree structure.

Trees as Models

- Trees are used as models in such diverse areas as computer science, chemistry, geology, botany, psychology, management, linguistics and many others. (See Epp,10.5[4ed], 10.4[5ed]).
- On your computer, the folder and file directory has a tree structure.
- Trees are used in linguistics to describe grammatical relationships.
- In botany and zoology, taxonomy of species is entirely based on a tree structure.
- In evolutionary science we have ‘the tree of life’.

Trees as Models

- Trees are used as models in such diverse areas as computer science, chemistry, geology, botany, psychology, management, linguistics and many others. (See Epp,10.5[4ed], 10.4[5ed]).
- On your computer, the folder and file directory has a tree structure.
- Trees are used in linguistics to describe grammatical relationships.
- In botany and zoology, taxonomy of species is entirely based on a tree structure.
- In evolutionary science we have ‘the tree of life’.
- Organic chemistry uses graphs and trees for models of molecules.

Trees as Models

- Trees are used as models in such diverse areas as computer science, chemistry, geology, botany, psychology, management, linguistics and many others. (See Epp,10.5[4ed], 10.4[5ed]).
- On your computer, the folder and file directory has a tree structure.
- Trees are used in linguistics to describe grammatical relationships.
- In botany and zoology, taxonomy of species is entirely based on a tree structure.
- In evolutionary science we have 'the tree of life'.
- Organic chemistry uses graphs and trees for models of molecules. We look at this next.

Structure of Hydrocarbon Molecules

- Graphs can be used to represent molecules, where atoms are represented by vertices and the bonds between them by edges.

Structure of Hydrocarbon Molecules

- Graphs can be used to represent molecules, where atoms are represented by vertices and the bonds between them by edges.
- **Hydrocarbon** molecules are composed of carbon and hydrogen only.

Structure of Hydrocarbon Molecules

- Graphs can be used to represent molecules, where atoms are represented by vertices and the bonds between them by edges.
- **Hydrocarbon** molecules are composed of carbon and hydrogen only.
- Each carbon atom normally forms 4 bonds with other atoms; we shall only consider this case.

Structure of Hydrocarbon Molecules

- Graphs can be used to represent molecules, where atoms are represented by vertices and the bonds between them by edges.

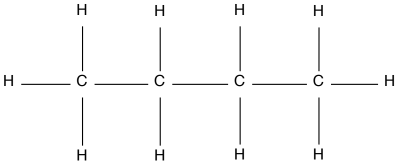
- **Hydrocarbon** molecules are composed of carbon and hydrogen only.
- Each carbon atom normally forms 4 bonds with other atoms; we shall only consider this case.
- Each hydrogen atom forms just one bond with another atom.

Structure of Hydrocarbon Molecules

- Graphs can be used to represent molecules, where atoms are represented by vertices and the bonds between them by edges.
- **Hydrocarbon** molecules are composed of carbon and hydrogen only.
- Each carbon atom normally forms 4 bonds with other atoms; we shall only consider this case.
- Each hydrogen atom forms just one bond with another atom.
- In the representation of such molecules we use C to represent a carbon atom and H to represent a hydrogen atom. These will be used instead of dots for the vertices.

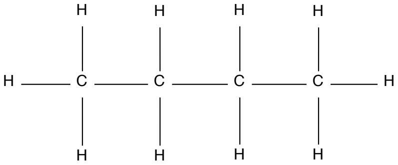
Examples of Hydrocarbon Graphs

- Butane:

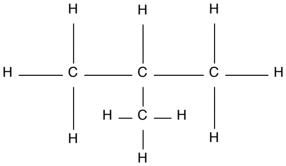


Examples of Hydrocarbon Graphs

- Butane:



- Isobutane:



Graphs and Isomers

- Butane and Isobutane graphs each have 4 carbon atoms and 10 hydrogen atoms, but the configuration of the atoms is different.

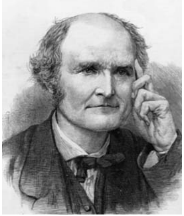
Graphs and Isomers

- Butane and Isobutane graphs each have 4 carbon atoms and 10 hydrogen atoms, but the configuration of the atoms is different.
- They have the same chemical formula, C_4H_{10}, but different chemical bonds and are called **isomers**.

Graphs and Isomers

- Butane and Isobutane graphs each have 4 carbon atoms and 10 hydrogen atoms, but the configuration of the atoms is different.
- They have the same chemical formula, C_4H_{10}, but different chemical bonds and are called **isomers**.
- **Saturated hydrocarbon** molecules contain the maximum number of hydrogen atoms for a given number of carbon atoms.

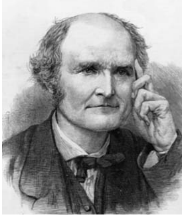
Some History of Trees



Arthur Cayley 1821 - 1895

- The English mathematician Arthur Cayley discovered trees when he was trying to enumerate the isomers of the saturated hydrocarbons of the form C_nH_{2n+2}.

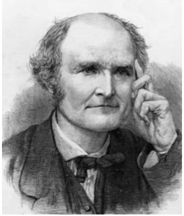
Some History of Trees



Arthur Cayley 1821 - 1895

- The English mathematician Arthur Cayley discovered trees when he was trying to enumerate the isomers of the saturated hydrocarbons of the form C_nH_{2n+2}.
- Cayley showed that a saturated hydrocarbon molecule must have this formula.

Some History of Trees



Arthur Cayley 1821 - 1895

- The English mathematician Arthur Cayley discovered trees when he was trying to enumerate the isomers of the saturated hydrocarbons of the form C_nH_{2n+2}.

- Cayley showed that a saturated hydrocarbon molecule must have this formula.
- You will explore a proof of this formula in Workshops next week.

Characterising Trees

Theorem: Let T be a graph with n vertices. The following statements are logically equivalent:

Characterising Trees

Theorem: Let T be a graph with n vertices. The following statements are logically equivalent:

(i) T is a tree (it is connected and has no non-trivial circuits).

Characterising Trees

Theorem: Let T be a graph with n vertices. The following statements are logically equivalent:

(i) T is a tree (it is connected and has no non-trivial circuits).

(ii) T has no simple circuits and $n - 1$ edges.

Characterising Trees

Theorem: Let T be a graph with n vertices. The following statements are logically equivalent:

(i) T is a tree (it is connected and has no non-trivial circuits).

(ii) T has no simple circuits and $n-1$ edges.

(iii) T is connected and has $n-1$ edges.

Characterising Trees

Theorem: Let T be a graph with n vertices. The following statements are logically equivalent:

(i) T is a tree (it is connected and has no non-trivial circuits).

(ii) T has no simple circuits and $n-1$ edges.

(iii) T is connected and has $n-1$ edges.

(iv) T is connected and every edge is a bridge.

Characterising Trees

Theorem: Let T be a graph with n vertices. The following statements are logically equivalent:

(i) T is a tree (it is connected and has no non-trivial circuits).

(ii) T has no simple circuits and $n-1$ edges.

(iii) T is connected and has $n-1$ edges.

(iv) T is connected and every edge is a bridge.

(v) Any two vertices of T are connected by exactly one simple path.

Characterising Trees

Theorem: Let T be a graph with n vertices. The following statements are logically equivalent:

(i) T is a tree (it is connected and has no non-trivial circuits).

(ii) T has no simple circuits and $n - 1$ edges.

(iii) T is connected and has $n - 1$ edges.

(iv) T is connected and every edge is a bridge.

(v) Any two vertices of T are connected by exactly one simple path.

(vi) T contains no non-trivial circuits, but the addition of any new edge (connecting an existing pair of vertices) creates a simple circuit.

Proving and Using the Theorem

- The previous Theorem collects together several different ways of characterizing a tree. In different contexts, each of the different descriptions are useful.

Proving and Using the Theorem

- The previous Theorem collects together several different ways of characterizing a tree. In different contexts, each of the different descriptions are useful.

- To prove the Theorem, we should show that any statement in the list is derivable from any other statement. One way to do this is to show the chain of implications:

$$(i) \Rightarrow (ii) \Rightarrow (iii) \Rightarrow (iv) \Rightarrow (v) \Rightarrow (vi) \Rightarrow (i).$$

Proving the Theorem

- *Can you prove that* $(i) \Rightarrow (j)$ *for any of the statements in the Tree-Characterising Theorem?*

Proving the Theorem

- *Can you prove that* $(i) \Rightarrow (j)$ *for any of the statements in the Tree-Characterising Theorem?*

- Some results that you might find helpful are ...

  Lemma A: Any tree that has more than one vertex has at least one vertex of degree 1.

Proving the Theorem

- *Can you prove that* $(i) \Rightarrow (j)$ *for any of the statements in the Tree-Characterising Theorem?*

- Some results that you might find helpful are ...

  Lemma A: Any tree that has more than one vertex has at least one vertex of degree 1.

  Lemma B: If G is any connected graph, C is a non-trivial circuit in G, and one of the edges of C is removed, then the subgraph that remains is connected.

Applications of graphs, paths and trees

- Many problems can be modelled with paths formed by travelling along the edges of graphs.

Applications of graphs, paths and trees

- Many problems can be modelled with paths formed by travelling along the edges of graphs.
- For instance, the problem of determining whether a message can be sent between two computers using intermediate links can be studied with a graph model.

Applications of graphs, paths and trees

- Many problems can be modelled with paths formed by travelling along the edges of graphs.
- For instance, the problem of determining whether a message can be sent between two computers using intermediate links can be studied with a graph model.
- Problems of efficiently planning routes for mail delivery, garbage pickup, diagnostics in computer networks, and so on, can be solved using models that involve graphs.

Applications of graphs, paths and trees

- Many problems can be modelled with paths formed by travelling along the edges of graphs.
- For instance, the problem of determining whether a message can be sent between two computers using intermediate links can be studied with a graph model.
- Problems of efficiently planning routes for mail delivery, garbage pickup, diagnostics in computer networks, and so on, can be solved using models that involve graphs.
- A subgraph of a connected graph that provides a unique path between any two vertices is a *spanning tree*. These trees have applications in many fields, including engineering.

Spanning trees

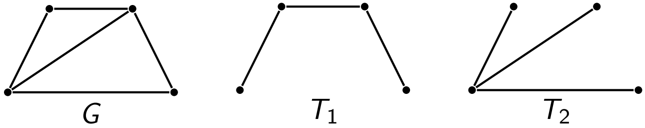
- **Definition:** A **spanning tree** for a graph G is a subgraph of G which is a tree and contains all the vertices of G.

Spanning trees

- **Definition:** A **spanning tree** for a graph G is a subgraph of G which is a tree and contains all the vertices of G.
- Spanning trees need not be unique. That is, a given graph can have a number of different spanning trees.

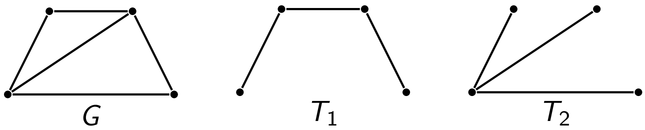
Spanning trees

- **Definition:** A **spanning tree** for a graph G is a subgraph of G which is a tree and contains all the vertices of G.
- Spanning trees need not be unique. That is, a given graph can have a number of different spanning trees.
- Example:Two distinct spanning trees, T_1 and T_2, for a graph G are shown below:



Spanning trees

- **Definition:** A **spanning tree** for a graph G is a subgraph of G which is a tree and contains all the vertices of G.
- Spanning trees need not be unique. That is, a given graph can have a number of different spanning trees.
- Example:Two distinct spanning trees, T_1 and T_2, for a graph G are shown below:



How many more spanning trees can you find for G?

Which graphs have spanning trees?

Theorem:

1. Every connected graph has a spanning tree.

Which graphs have spanning trees?

Theorem:

1. Every connected graph has a spanning tree.

   Can you prove this?

Which graphs have spanning trees?

Theorem:

1. Every connected graph has a spanning tree.

   Can you prove this?

   Hint: Use Lemma B.

Which graphs have spanning trees?

Theorem:

1. Every connected graph has a spanning tree.

   Can you prove this?

   Hint: Use Lemma B.

2. Any 2 spanning trees for a graph have the same number of edges.

Which graphs have spanning trees?

Theorem:

1. Every connected graph has a spanning tree.

   Can you prove this?

   Hint: Use Lemma B.

2. Any 2 spanning trees for a graph have the same number of edges.

   Why?

Which graphs have spanning trees?

Theorem:

1. Every connected graph has a spanning tree.

   Can you prove this?

   Hint: Use Lemma B.

2. Any 2 spanning trees for a graph have the same number of edges.

   Why?

   Hint: See (ii) or (iii) of the tree characterisation theorem.

A method to make a spanning tree

Let G be a connected graph with n vertices.

A method to make a spanning tree

Let G be a connected graph with n vertices.

The Algorithm

1. Initialize T to be the vertices of G but no edges.

A method to make a spanning tree

Let G be a connected graph with n vertices.

The Algorithm

1. Initialize T to be the vertices of G but no edges.
2. Pick an edge.

A method to make a spanning tree

Let G be a connected graph with n vertices.

The Algorithm

1. Initialize T to be the vertices of G but no edges.
2. Pick an edge.
3. Add the chosen edge to T if and only if it does not make a circuit in T.

A method to make a spanning tree

Let G be a connected graph with n vertices.

<u>The Algorithm</u>

1. Initialize T to be the vertices of G but no edges.
2. Pick an edge.
3. Add the chosen edge to T if and only if it does not make a circuit in T.
4. Repeat steps (2) and (3) until T has $n-1$ edges, then stop.

A method to make a spanning tree

Let G be a connected graph with n vertices.

The Algorithm

1. Initialize T to be the vertices of G but no edges.
2. Pick an edge.
3. Add the chosen edge to T if and only if it does not make a circuit in T.
4. Repeat steps (2) and (3) until T has $n-1$ edges, then stop.

The above description of the algorithm is intended for ‘pen-and-paper’ work.

A method to make a spanning tree

Let G be a connected graph with n vertices.

The Algorithm

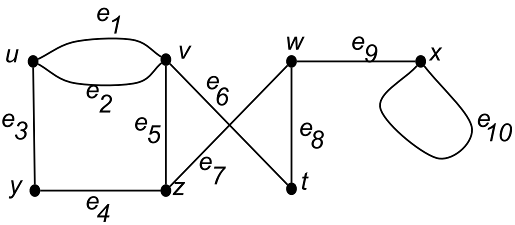
1. Initialize T to be the vertices of G but no edges.
2. Pick an edge.
3. Add the chosen edge to T if and only if it does not make a circuit in T.
4. Repeat steps (2) and (3) until T has $n-1$ edges, then stop.

The above description of the algorithm is intended for 'pen-and-paper' work.

For computer implementation it is necessary to **also**:
— at step 1 initialize a 'pool of potential edges' P to $E(G)$,
— at step 2 ensure the picked edge e comes from P and
— after step 3 remove e from P (whether it contributes to T or not).

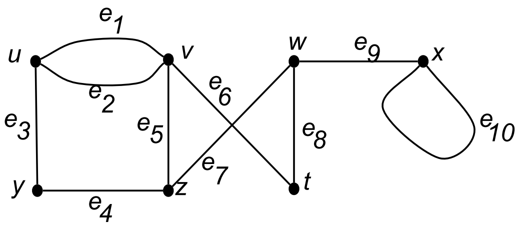
Building a spanning tree: example

To demonstrate the spanning tree algorithm we will use a graph we have seen before:



Building a spanning tree: example

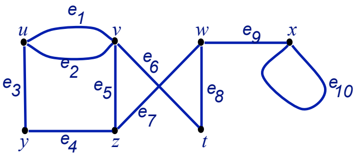
To demonstrate the spanning tree algorithm we will use a graph we have seen before:



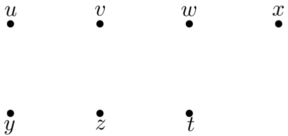
Initialize T to

- vertex set $V(T) = \{u, v, w, x, y, z, t\}$,
- edge set $E(T) = \{\}$:

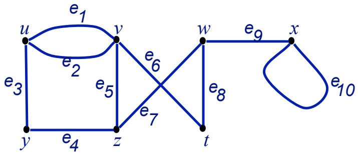
Building a spanning tree for



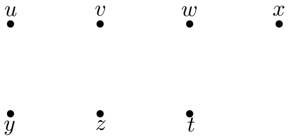
$E(T) = \{\}$: $\quad T$:



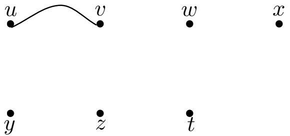
Building a spanning tree for



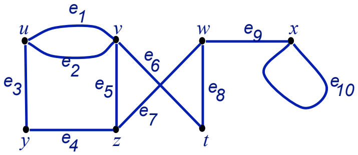
$E(T) = \{\}$: $T:$



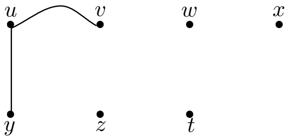
$E(T) = \{e_1\}$ $T:$



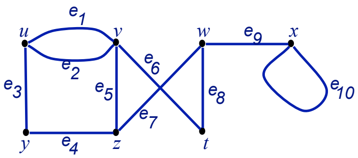
Building a spanning tree for



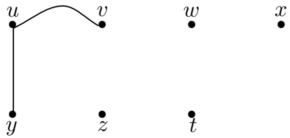
$E(T) = \{e_1, e_3\}$: T :



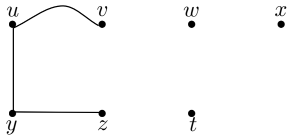
Building a spanning tree for



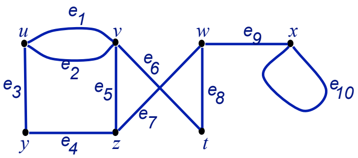
$E(T) = \{e_1, e_3\}$: $T:$



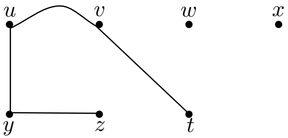
$E(T) = \{e_1, e_3, e_4\}$ $T:$



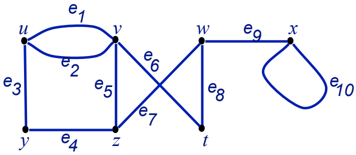
Building a spanning tree for



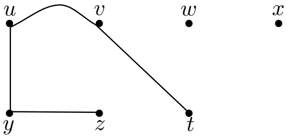
$E(T) = \{e_1, e_3, e_4, e_6\}$: $\quad T:$



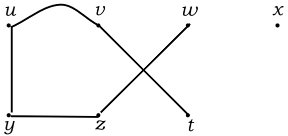
Building a spanning tree for



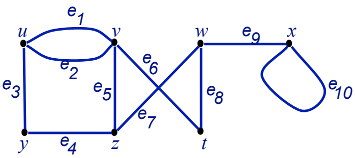
$E(T) = \{e_1, e_3, e_4, e_6\}$: T :



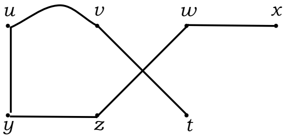
$E(T) = \{e_1, e_3, e_4, e_6, e_7\}$ T :



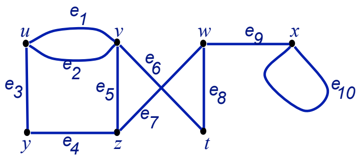
Completed spanning tree for



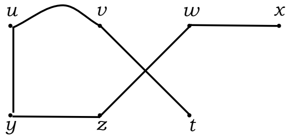
$E(T) = \{e_1, e_3, e_4, e_6, e_7, e_9\}$: T :



Completed spanning tree for

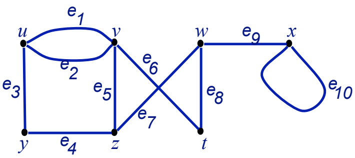


$E(T) = \{e_1, e_3, e_4, e_6, e_7, e_9\}$: T :



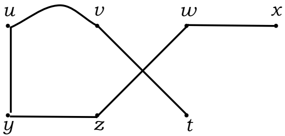
Our tree now has $7 - 1 = 6$ edges, so we are done (as you can see).

Completed spanning tree for



$E(T) = \{e_1, e_3, e_4, e_6, e_7, e_9\}$:

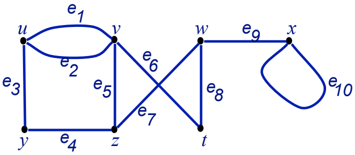
T :



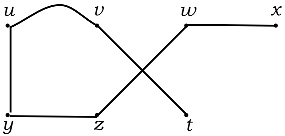
Our tree now has $7 - 1 = 6$ edges, so we are done (as you can see).

As it happens, this tree has no branching and so contains a path of length 6. That just results from our order of choosing edges.

Completed spanning tree for



$E(T) = \{e_1, e_3, e_4, e_6, e_7, e_9\}$: T :

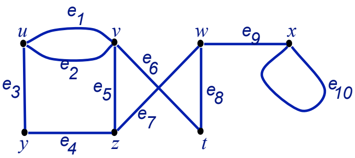


Our tree now has $7 - 1 = 6$ edges, so we are done (as you can see).

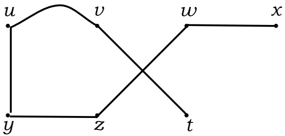
As it happens, this tree has no branching and so contains a path of length 6. That just results from our order of choosing edges.

Can you make a spanning tree in which the longest path has length 4?

Completed spanning tree for



$E(T) = \{e_1, e_3, e_4, e_6, e_7, e_9\}$: T :

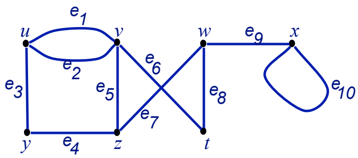


Our tree now has $7 - 1 = 6$ edges, so we are done (as you can see).

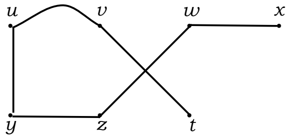
As it happens, this tree has no branching and so contains a path of length 6. That just results from our order of choosing edges.

Can you make a spanning tree in which the longest path has length 4? Length 3?

Completed spanning tree for



$E(T) = \{e_1, e_3, e_4, e_6, e_7, e_9\}$: T :



Our tree now has $7 - 1 = 6$ edges, so we are done (as you can see).

As it happens, this tree has no branching and so contains a path of length 6. That just results from our order of choosing edges.

Can you make a spanning tree in which the longest path has length 4? Length 3?

END OF SECTION D1