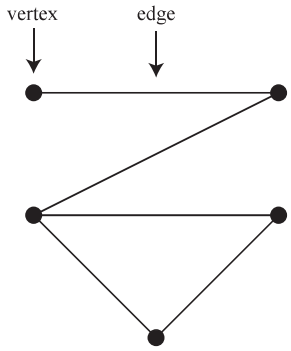
Part A Graph Theory

Marc Lackenby
With minor edits by Richard Earl

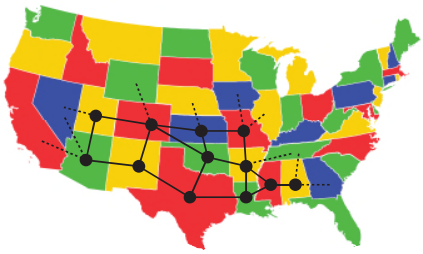
Trinity Term 2024

1 Introduction and Definitions

Graph theory is the mathematical theory of networks. A graph has 'nodes' called *vertices* connected by 'lines' called *edges*. Graphs may be used to describe a wide range of physical, biological and social systems, diversely representing transport networks (such as the London underground), computer networks and website structure, evolution of words across languages and time in philology, migrations in biology and geography, etc.. The definition of a graph may be extended to include one-way edges; such graphs are known as *directed graphs* or *digraphs*. We might also introduce weights to edges in a *weighted graph* to demonstrate the difficulty – say in terms of time, distance or cost – of travelling along a particular edge. Formal definitions will follow shortly.

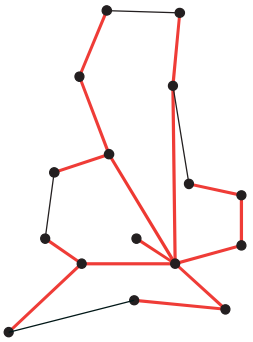


A famous result in graph theory is the *Four Colour Theorem*. This answers a question first posed by Francis Guthrie in 1852: Is it possible to colour countries on a map using only four colours, so that any two countries sharing a border receive different colours?



This was proved by Appel and Haken in 1976, using a controversial computer-assisted proof. (Intriguingly this problem in the plane proved the last and hardest case; the problem for closed surfaces of positive genus, such as a torus, was solved by 1968. On a torus up to 7 colours are needed.) Note how the problem can be represented with a graph; each country is a vertex and 'sharing a border' becomes an edge.

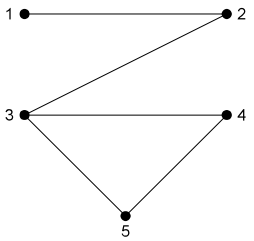
Another example: say the government wants to build a new high speed rail network that links all of the major cities in the country. It wants to decide which existing rail lines to upgrade. The government's main priority is not to minimise journey times, but rather to minimise the cost subject to making a connected network.



The red and black lines together represent all current rail links; the red lines represent a *connected* upgraded network where it's still possible to travel between all cities (vertices). In such an example, the edges would need a *cost* or *weight* associated with each.

Let's make some definitions and formulate these problems mathematically.

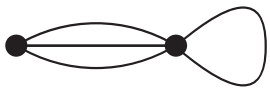
Definition 1. *A graph $G = (V(G), E(G))$ consists of two sets: $V(G)$ is the **vertex set** and $E(G)$ is the **edge set**, where each element of $E(G)$ consists of an unordered pair of elements of $V(G)$. We will always assume, without further comment, that $V(G)$ is finite.*



For the above graph G we have that $V(G) = \{1, 2, 3, 4, 5\}$ and $E(G) = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{3, 5\}\}$.

Note that we use the term 'graph' where some would say 'simple graph', using the term 'graph' for a more general structure which allows several edges between a given pair of vertices and also 'loops' – edges that join a vertex to itself. To distinguish such 'graphs' from simple graphs, they are often referred to as 'multigraphs'.

There is also the notion of a *directed* graph or *digraph*, where travel along an edge (or *arc*) may only be permitted in one direction. Such digraphs can model one-way systems and systems where a certain process might not be straightforwardly reversible, or not at all. Or a weighted directed graph might recognize that the cost of moving along an edge in one direction is distinct to that in the opposite direction – for example, when going up or down a hill. **In this course, though, we will only consider simple graphs – which we just refer to as 'graphs' and simple graphs with weights.**



a non-simple multigraph

We write $uv = \{u, v\} = vu$ for the (unordered) pair representing an edge between the vertices u and v.

Definition 2. *Let G be a graph.*

*(a) A **walk** in G is a sequence W of vertices $v_1, \ldots, v_t$ such that $v_i v_{i+1} \in E(G)$ for each $1 \leqslant i < t$. We say that the **length** of W is $t - 1$.*

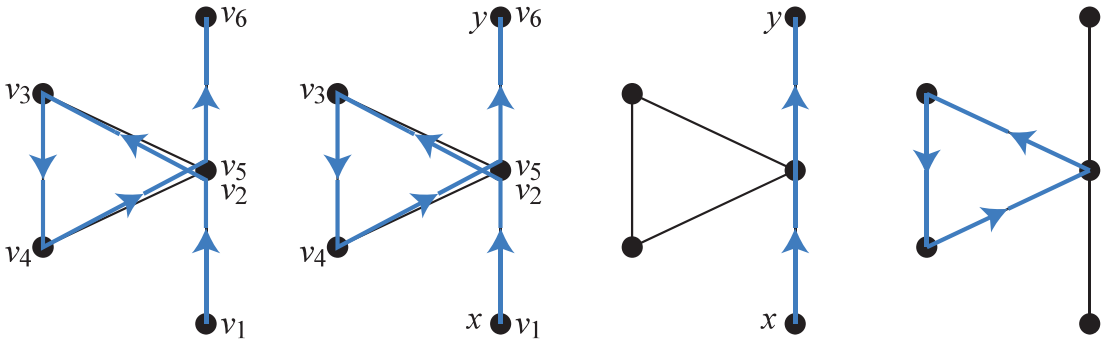
*(b) If we want to specify that start and end of the walk, then we call W an xy-**walk** with $x = v_1$ and $y = v_t$.*

*(c) If the vertices in W are distinct we call it a **path**.*

*(d) If $x = y$ we call W a **closed** walk.*

*(e) If $x = y$, but the vertices are otherwise distinct, and W has at least 3 vertices, then we call W a **cycle**.*

(f) We also regard paths and cycles (but not walks, as vertices may be repeated) as subgraphs of G.



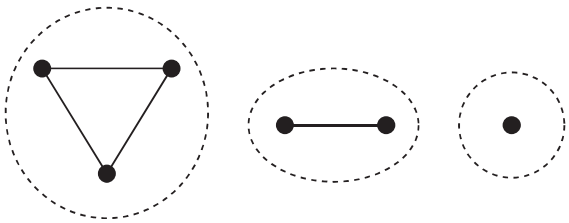
a walk, an xy-walk, an xy-path, a cycle

Definition 3.

*(a) We say that G is **connected** if for any $x, y \in V(G)$ there is an xy-walk in G.*

*(b) We say that two vertices x and y of a graph G lie in the same **component** if they are joined by an xy-walk.*

Clearly being in the same component is an equivalence relation, and so the components partition $V(G)$.



the three components of a disconnected graph

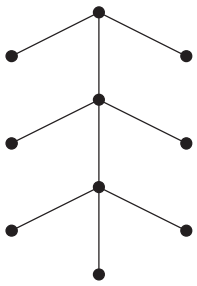
A natural question at this point is: What can we say about the possible $S \subseteq E(G)$ such that $(V(G), S)$ is connected? Such a question would be important to the earlier rail network problem. We say that $(V(G), S)$ is *minimally connected* if $(V(G), S)$ is connected but $(V(G), S \setminus \{e\})$ is not connected for any $e \in S$. This motivates the next section.

2 Trees

Definition 4.

*(a) A **tree** is a minimally connected graph.*

*(b) If a graph G has no cycle we call it **acyclic**.*

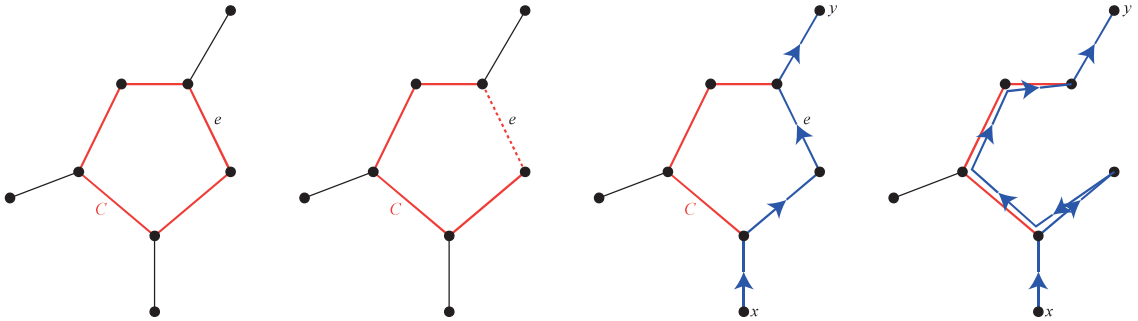


Lemma 5. *Any tree is acyclic.*

Proof. Let G be a tree. Suppose for a contradiction that G contains a cycle C and let $e \in E(C)$. We will obtain our contradiction by showing that

$$G - e := (V(G), E(G) \setminus \{e\})$$

is connected. Let P be the path obtained by deleting e from C. Consider any x, y in $V(G)$. As G is connected, there is an xy-walk W in G. Replacing any use of e in W by P gives an xy-walk in $G - e$. Thus $G - e$ is connected, contradiction. □

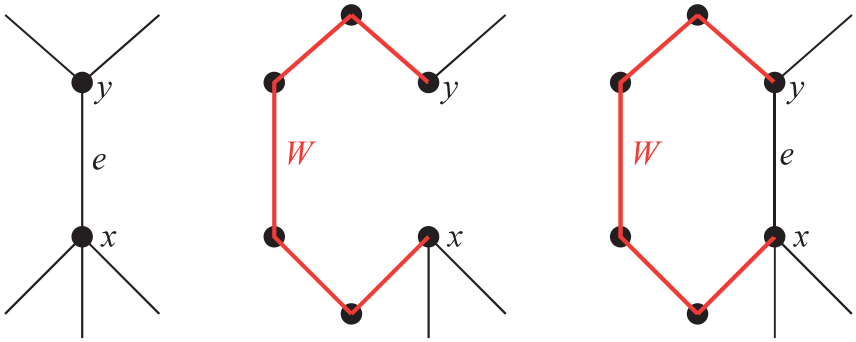


There are many equivalent characterisations of trees, any of which could be taken as the definition. Here is one:

Lemma 6. *G is a tree if and only if G is connected and acyclic.*

Proof. ($\Rightarrow$) If G is a tree then G is connected by definition and acyclic by Lemma 5.

($\Leftarrow$) Conversely, let G be connected and acyclic. Suppose, for a contradiction, that $G - e$ is connected for some $e = xy \in E(G)$. Let W be a shortest xy-walk in $G - e$. Then W must be a path, i.e. W has no repeated vertices, otherwise we would find a shorter walk by deleting a segment of W between two visits to the same vertex. Combining W with xy gives a cycle, contradiction. □



The fact that a shortest walk between two points is a path is often useful. More generally, considering an extremal (shortest, longest, minimal, maximal, ...) object is often a useful proof technique. Another example:

Lemma 7. *Any two vertices in a tree are joined by a unique path.*

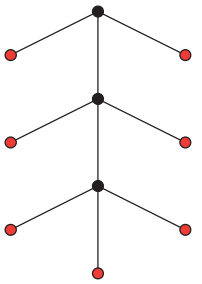
Proof. Suppose for a contradiction that this fails for some tree G. Choose x, y in $V(G)$, so that there are distinct xy-paths P_1, P_2, and P_1 is as short as possible over all such choices of x and y. Then P_1 and P_2 only intersect in x and y (or else distinct paths between one end and an intersection would exist, contradicting the minimality of P_1). So the union of P_1 and P_2 is a cycle, contradicting Lemma 6. □

Definition 8. *Let G be a graph.*

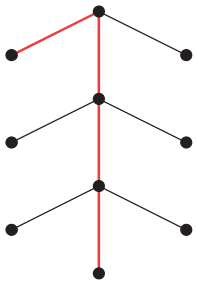
*(a) If $uv \in E(G)$ we say that u and v are **neighbours** and are **adjacent** vertices.*
*(b) If $e = uv \in E(G)$ we say the edge e is **incident** to u and v.*
*(c) The **degree** $d(v)$ of a vertex v is the number of neighbours of v in G.*
*(d) A **leaf** is a vertex of degree one, i.e. a vertex with a unique neighbour.*

Lemma 9. *Any tree with at least two vertices has at least two leaves.*

Proof. Consider any tree G. Let P be a longest path in G. The two ends of P must be leaves of G. Indeed, an end cannot have a neighbour in $V(G) \setminus V(P)$, or we could make P longer, and cannot have any neighbour in $V(P)$ other than the next in the sequence of P, or we would have a cycle. □



leaves of a tree,



a longest path

Lemma 10. *If G is a tree and v is a leaf of G, then $G - v$ is a tree.*

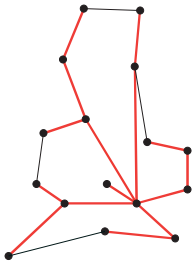
Proof. Note we define $V(G - v) = V(G) \setminus \{v\}$ and $E(G - v) = \{xy \in E(G) \mid v \notin \{x, y\}\}$.

By Lemma 6 it suffices to show that $G - v$ is connected and acyclic. Acyclicity is immediate from Lemma 6. Connectivity follows by noting for any x, y in $V(G) \setminus \{v\}$ that the unique xy-path in G is contained in $G - v$. □

Lemma 11. *Any tree on n vertices has $n - 1$ edges.*

Proof. By induction on the number of vertices. A tree with 1 vertex has 0 edges. Let G be a tree on $n > 1$ vertices. By Lemma 9, G has a leaf v. By Lemma 10, $G - v$ is a tree. By the inductive hypothesis, $G - v$ has $n - 2$ edges. Replacing v gives $n - 1$ edges in G. □

Definition 12. *Any connected graph G contains a minimally connected subgraph (i.e. a tree) with the same vertex set, which we call a **spanning tree** of G.*



Lemma 13. *A graph G is a tree on n vertices if and only if G is connected and has $n - 1$ edges.*

Proof. If G is a tree then G is connected by definition and has $n - 1$ edges by Lemma 11.

Conversely, suppose that G is connected and has $n-1$ edges. Let H be a spanning tree of G. Then H has $n-1$ edges by Lemma 11, so $H = G$, so G is a tree. □

3 MCSTs

Recall, from the first section, the *high-speed network question*: let G be a connected graph such that for every edge $e \in E(G)$ we are given a *cost* or *weight* $c(e) > 0$. More generally, for any $S \subseteq E(G)$ we set

$$c(S) = \sum_{e \in S} c(e)$$

as the cost of S. Our task is:

Find $S \subseteq E(G)$ with minimum possible $c(S)$
such that $(V(G), S)$ is a connected graph.

Such an S would necessarily be minimally connected, as the cost of each edge is positive, and so would be a tree – specifically a **minimum cost spanning tree** or **MCST**.

A very inefficient algorithm that solves the task is the following: list all $S \subseteq E(G)$, check each one to see whether $(V(G), S)$ is a connected graph, compute $c(S)$ for each, and take the cheapest one. This would be unwise because there are $2^{|E(G)|}$ subsets of $E(G)$, so we could never check them all in practice unless G is very

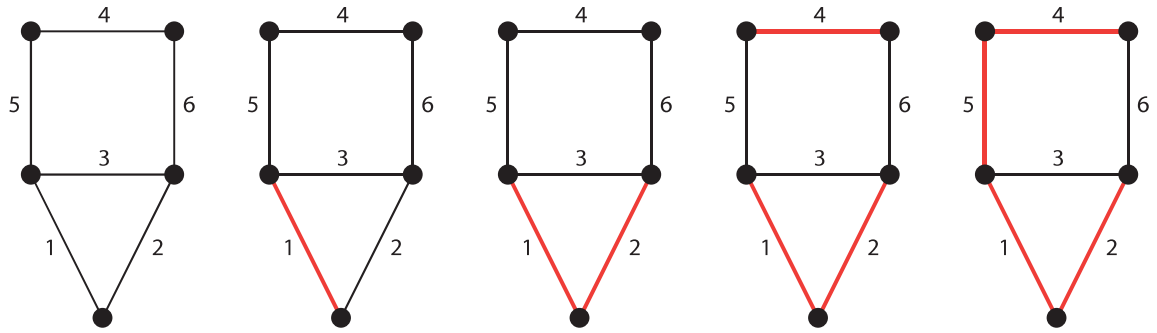
small. We are interested in *efficient* algorithms which run in *polynomial time* - that is, some polynomial in $|V(G)|$ and $|E(G)|$ bounds the number of operations executed.

By contrast, an efficient algorithm is **Kruskal's algorithm** (1956), described below.

At step $i \geqslant 0$, we will keep track of a subset $A_i \subseteq E(G)$. This will have the property that $(V(G), A_i)$ is acyclic.

Start with $A_0 = \emptyset$. At step $i \geqslant 0$, is there an edge $e \in E(G) \setminus A_i$ such that $(V(G), A_i \cup \{e\})$ is acyclic?

If no, then output $A = A_i$ and stop.
If yes, then set $A_{i+1} = A_i \cup \{e\}$ for one such e such that $c(e)$ is minimal, and proceed to step $i + 1$.

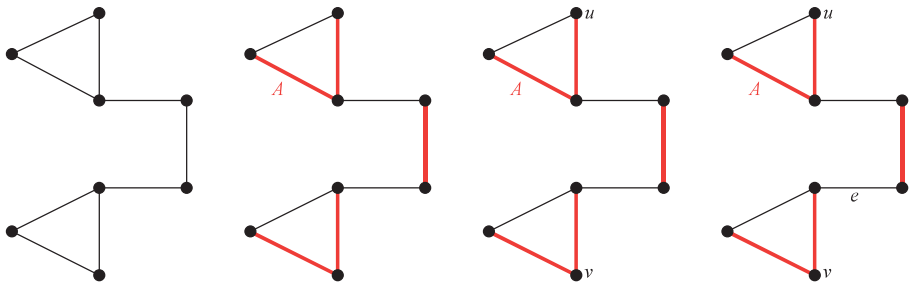


A_0 A_1 A_2 A_3 A_4

Note that we do not include edge 3 in A_4 as this would create a (triangular) cycle.

Theorem 14. *$(V(G), A)$ is a minimum cost spanning tree of G.*

Proof. Firstly, we will prove that $(V(G), A)$ is a spanning tree of G. By construction, it is acyclic. Suppose, for a contradiction, that $(V(G), A)$ is not connected. Let u, v lie in distinct components of $(V(G), A)$. As G is connected, there is a uv-walk. This must contain an edge e of G whose endpoints are in different components. So $A \cup \{e\}$ is acyclic and the algorithm should not have terminated when it did. Instead, it should have added e to A_i. This contradiction shows that $(V(G), A)$ is a spanning tree of G.



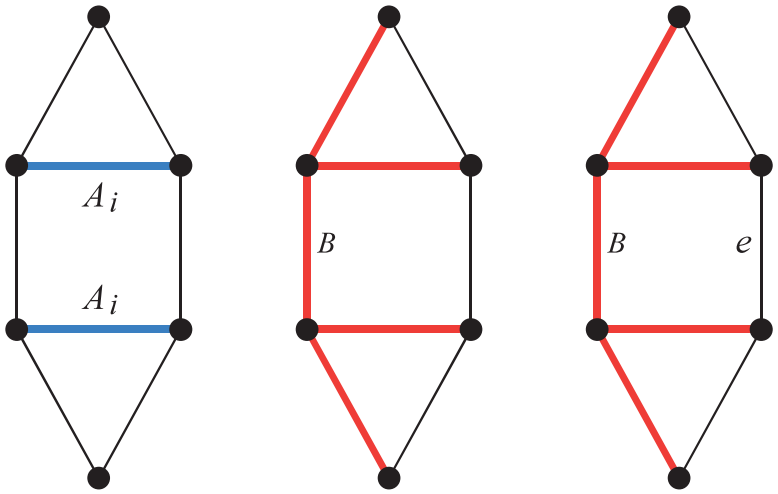
We now prove that $(V(G), A)$ has minimum cost. Let $\mathcal{M}$ be the set of $B \subset E(G)$ such that $(V(G), B)$ is a MCST. We will prove by induction on i that

$$(*) \quad \text{there is a } B \in \mathcal{M} \text{ with } A_i \subseteq B.$$

Note that $(*)$ will suffice to prove the theorem, as when we apply it to $A_i = A$ we will have $A \subseteq B$ for some $B \in \mathcal{M}$ and so $|A| = |B|$ by Lemma 13, and so $A = B \in \mathcal{M}$.

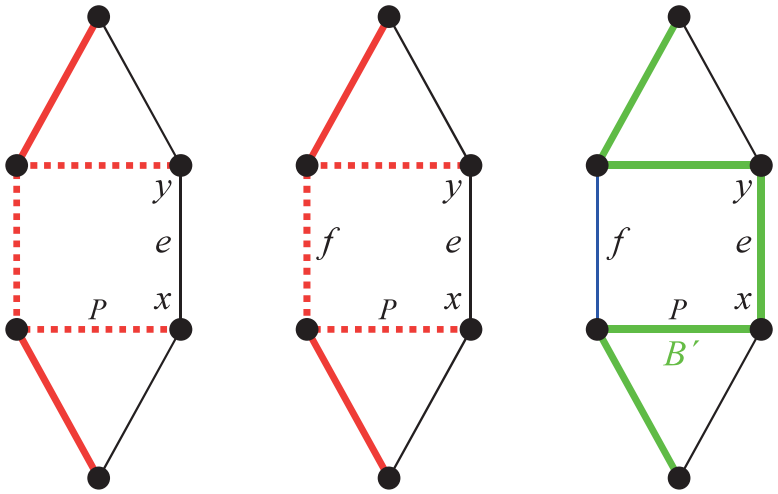
Proving $(*)$: Base case $i = 0$ of $(*)$. We have $A_0 = \emptyset$, so any $B \in \mathcal{M}$ satisfies $(*)$.

Inductive step. Suppose for some $i \geqslant 0$, we have $A_i \subseteq B \in \mathcal{M}$. We can suppose $A_i \neq A$, otherwise the proof is complete. Consider $A_{i+1} = A_i \cup \{e\}$ given by the algorithm. We need to find $B' \in \mathcal{M}$ with $A_{i+1} \subseteq B'$. We can assume $e \notin B$, otherwise we could take $B' = B$.



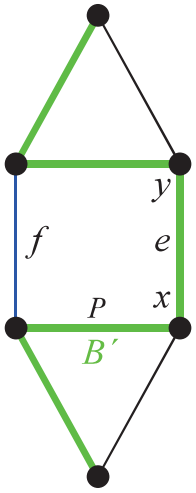
Let $e = xy$ and let P be the unique xy-path in the spanning tree $(V(G), B)$. Then $C = P \cup \{e\}$ is a cycle. As A_{i+1} is acyclic, we can choose $f \in C \setminus A_{i+1}$. Let $B' = (B \setminus \{f\}) \cup \{e\}$. To finish the proof we need to show that

1. $A_{i+1} \subseteq B'$,
2. $(V(G), B')$ is a spanning tree, and
3. $c(B') \leqslant c(B)$.



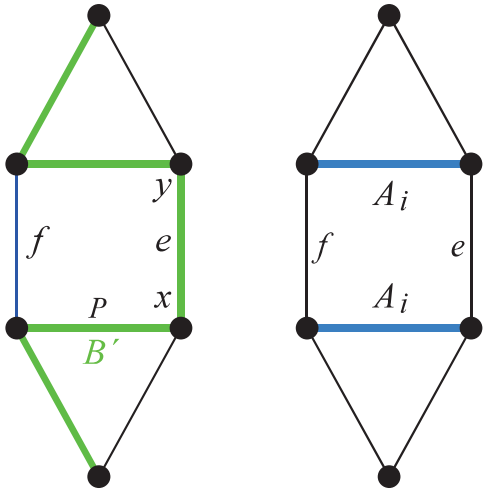
Showing $A_{i+1} \subseteq B'$: Note that $A_{i+1} = A_i \cup \{e\} \subseteq B'$, as $A_i \subseteq B$ and $f \notin A_{i+1}$.

Showing $(V(G), B')$ is a spanning tree: Note that B' is connected, for the following reason. Any two vertices in $V(G)$ are joined by a path in B. Replace each occurence of f in this path by $C \setminus \{f\}$. Also B' has $|V(G)| - 1$ edges. So it is a spanning tree by Lemma 13.



Showing $c(B') \leqslant c(B)$: Note that $A_i \cup \{f\} \subseteq B$, so $A_i \cup \{f\}$ is acyclic. Now e was chosen so that $c(e)$ is minimal among all edges e such that $A_i \cup \{e\}$ is acyclic. Hence, $c(e) \leqslant c(f)$. So $c(B') = c(B) - c(f) + c(e) \leqslant c(B)$.

This finishes the proof of the inductive step of $(*)$, and so of the theorem.



□

We consider now how fast Kruskal's algorithm is. Making this question mathematically precise would take us far afield (we would need to define a model of computation). Instead, we will take an intuitive approach, estimating the number of 'steps' taken by an algorithm, where a 'step' should be a 'simple' operation.

Recall that in Kruskal's algorithm:

With each iteration we add an edge, so there are $|V(G)| - 1$ iterations.

If at each stage of the algorithm, we naively find the next edge by checking every edge then there will be $|E(G)|$ steps in each iteration, giving about $|V(G)||E(G)|$ steps in total.

We say that the running time is $O(|V(G)||E(G)|)$, where the 'big O' notation means that there a positive constant C so that for any graph G the running time is at most $C|V(G)||E(G)|$.

Here 'running time' could be measured in any units, say milliseconds on your favourite computer, as changing the units or using a different computer will just replace C by a different constant.

A smarter implementation is to start by making a list of all edges ordered by cost, cheapest first. Then at each step we go through the list from the start, discarding edges that make a cycle until we find the first edge which can be added.

This gives a running time that is 'roughly comparable' with the number of edges, which is essentially the best possible.

4 Euler tours

The town of Königsberg (now Kaliningrad), on the Baltic Sea, iss divided into 4 districts by the river Pregel.

In the 18th century, the river was spanned by 7 bridges and at the time the following question was unanswered:

Is it possible to take a walk that crosses every bridge exactly once?

The Königsberg bridge problem was solved by Leonhard Euler in 1736. It was the first ever result in graph theory (and arguably the first in topology).

Definition 15. *Let W be a walk in a graph G.*
*(a) We call W an **Euler trail** if every edge of G appears exactly once in W.*
*(b) An **Euler tour** (or an **Euler circuit**) is a closed Euler trail, i.e. it starts and ends at the same vertex.*
*(c) A graph with an Euler tour is said to be **Eulerian**.*

Here we will only solve the problem of finding an Euler tour; the solution of the Euler trail problem can be deduced (exercise sheet 1, question 7).

Clearly, an Eulerian graph must be connected after we delete any *isolated* vertices (i.e. vertices of degree zero). Next we note that each visit of an Euler tour W to a vertex v uses two edges at v (one to arrive and one to leave). This is also true of the start and end vertex of W. As every edge is used exactly once, we deduce that every vertex has even degree; this then is a necessary condition for a connected graph to be Eulerian. The Königsberg bridges are certainly not Eulerian as the vertices have degrees 3, 3, 3, 5. It is not hard to see that there is no Euler trail either.

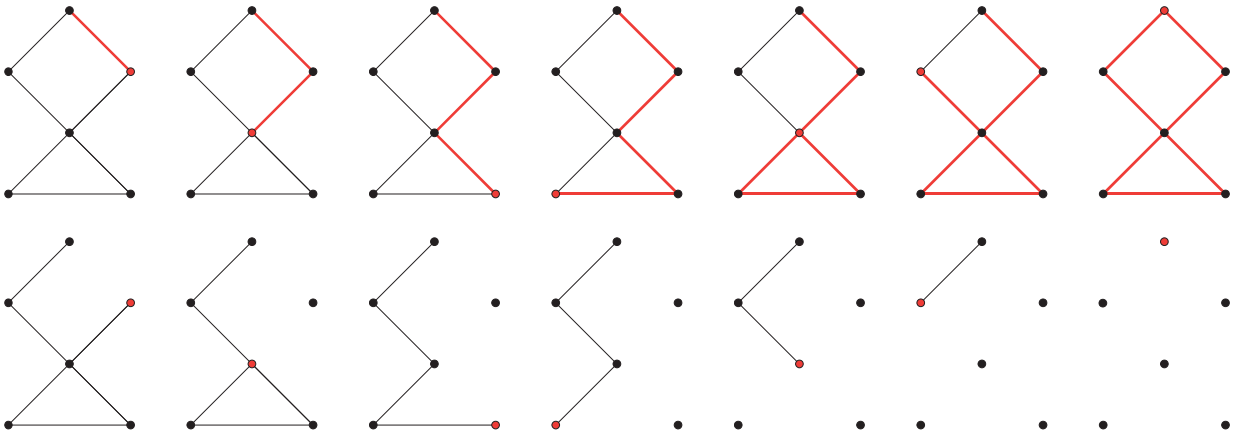
Theorem 16. *A connected graph is Eulerian if and only if each vertex has even degree.*

In fact, we will show that in such a case we can efficiently find an Euler tour, using *Fleury's algorithm.*

Start at any vertex of G. We will follow a walk, erasing each edge after it is used (erased edges cannot be used again). At each stage, ensure that the following holds:

1. when the edge is removed, the resulting graph is connected once isolated vertices are removed, and
2. we do not run along an edge to a leaf, unless this is the only remaining edge of the graph.

We will show that when each vertex has even degree, this algorithm produces an Euler tour. The running time of Fleury's algorithm is $O(|E(G)|^2)$, so it is technically efficient, but there are better algorithms that run in $O(|E(G)|)$ steps.



Fleury's Algorithm (1883)

Steps: 1 2 3 4 5 6 7

We first prove the *handshaking lemma*.

Lemma 17. *In any graph, there are an even number of vertices with odd degree.*

Proof. Since every edge has two endpoints,

$$\sum_{v \in V(G)} d(v) = 2|E(G)|.$$

(This is known as a the *degree sum formula*.) Therefore, in the sum, there must be an even number of occurrences of $d(v)$ for which $d(v)$ is odd. □

Proof. (Of Fleury's algorithm.) We now show that Fleury's algorithm yields an Euler tour. Note firstly, that at each stage of the algorithm, either there are two vertices of odd degree (the initial vertex u and the current one) or there are no vertices of odd degree.

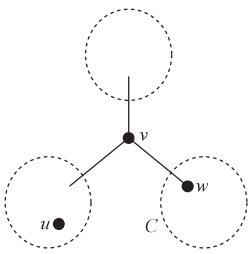
Suppose, for a contradiction, that Fleury's Algorithm fails. Say it stops at some vertex v and can go no further. Let H be the component of the current graph containing v.

The degree of v in H must be positive, as otherwise in the previous step, we ran along an edge to a leaf violating condition (2). If the degree of v in H is one, then we can continue the walk. So there are at least two edges of H containing v.

Since the algorithm cannot continue, the graph $H - e$ is disconnected for each edge e containing v. Hence, the edges e incident to v all have endpoints in distinct components of $H - v$.

So, we can choose one edge vw, such that the component C of $G - vw$ which contains w does not contain the first vertex u of the walk. But then w is the only vertex of odd degree in C, which is impossible by the handshaking lemma.

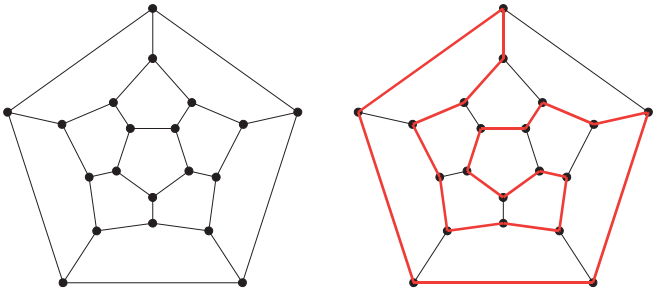
□



5 Hamiltonian cycles

Instead of focusing on edges, we can instead pose a similar question about a connected graph:

Does there exists a closed walk that visits every vertex exactly once?



Such a walk is necessarily a cycle and is known as a *Hamiltonian cycle*. When a graph contains such a cycle, it is said to be *Hamiltonian*.

Unlike the case of Eulerian tours, it turns out that there is, almost certainly, no efficient algorithm to determine whether a general graph is Hamiltonian. By 'efficient', we mean that the algorithm provides the answer after polynomially many 'steps', as a function of $|V(G)|$ and $|E(G)|$. Mathematicians currently do not have a proof that there is no efficient algorithm to determine whether a general graph is Hamiltonian. We say 'almost certainly' because *it is known* that if the famous conjecture (and Millennium Problem) $P \neq NP$ is true, then there is no efficient algorithm to find Hamiltonian cycles. But to discuss this conjecture in any detail would take us too far afield.

We will therefore content ourselves with a sufficient condition for a graph to be Hamiltonian.

Theorem 18. ***(Ore's Theorem** (1960))*
Let G be a connected graph with $n \geqslant 3$ vertices. Suppose that for every pair of non-adjacent vertices x and y,

$$d(x) + d(y) \geqslant n.$$

Then G is Hamiltonian.

Corollary 19. ***(Dirac's Theorem** (1952)) [Gabriel Dirac was Paul Dirac's adopted son.]*
If G is connected with $n \geqslant 3$ vertices and for every vertex v, $d(v) \geqslant n/2$, then G is Hamiltonian.

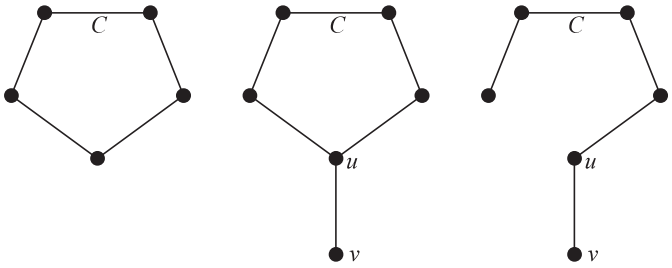
As an example the complete graph on n vertices, K_n can easily be seen to be Hamiltonian. This is the graph with n vertices where there is an edge between each pair of distinct vertices. Ore's theorem immediately applies, vacuously, as there are no non-adjacent vertices. But it's much easier to describe a Hamiltonian cycle namely $v_1v_2v_3 \dots v_nv_1$.

The complete bipartite graph on m and n vertices, $K_{m,n}$ is not Hamiltonian when $m \neq n$ and is Hamiltonian when $m = n$. This is the graph where there is an edge between every vertex of one part and every vertex of the other part (and no further edges).

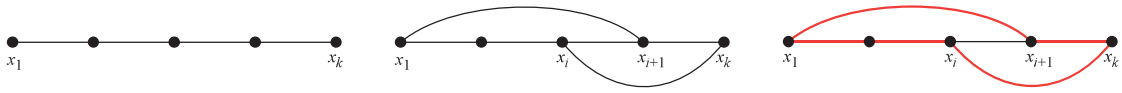
We first note if G is Hamiltonian and has n vertices, then the length of the longest cycle is n and the length of the longest path is $n - 1$. That is, there are n edges in the longest cycle and $n - 1$ edges in the longest path.

Lemma 20. *If G is connected and non-Hamiltonian, then the length of the longest path is at least the length of the longest cycle.*

Proof. Let C be a longest cycle, with length ℓ. Since G is non-Hamiltonian, there is some vertex not in C. Since G is connected, there is therefore some edge uv with one endpoint u in C and one endpoint v not in C. Removing an edge of C incident to u and adding uv gives a path of length ℓ. □



Proof. (Of Ore's Theorem)
Suppose that G is not Hamiltonian. Let $P = x_1 \cdots x_k$ be a longest path. It has length $k-1 < n$ as G is Hamiltonian.



So by Lemma 20, G does not have a cycle of length k. So x_1 and x_k are not adjacent. Hence, by our assumption, $d(x_1) + d(x_k) \geqslant n$. There is no integer i such that x_1 is adjacent to x_{i+1} and x_k is adjacent x_i. Otherwise, $x_1 \cdots x_i x_k x_{k-1} \cdots x_{i+1} x_1$ would be a cycle of length k. So the sets

$$A = \{i \mid x_1 x_{i+1} \in E(G)\}, \qquad B = \{i \mid x_i x_k \in E(G)\}$$

are disjoint subsets of $\{1, \cdots, k-1\}$.

By the proof of Lemma 9, as P is a longest path, every neighbour of x_1 lies in P, and similarly every neighbour of x_k lies in P. So, A has size $d(x_1)$, and B has size $d(x_k)$. Since A and B are disjoint, $d(x_1)+d(x_k) \leqslant k-1 < n$, which is a contradiction. Hence, G must be Hamiltonian.

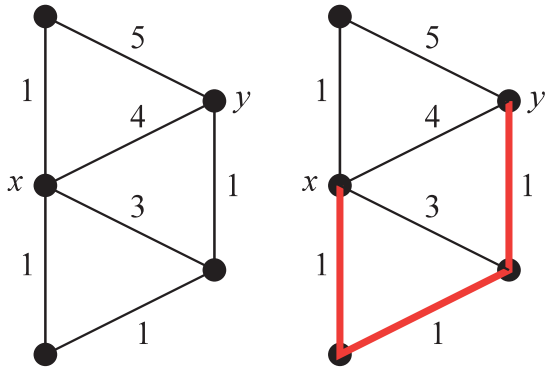
□

6 Shortest Paths

Let G be a connected graph, with $\ell(e) > 0$ an assigned length' for each edge $e \in E(G)$. The *ℓ-length* of a path P is

$$\ell(P) = \sum_{e \in E(P)} \ell(e).$$

Given x and y in $V(G)$, an *ℓ-shortest xy-path* is an xy-path P that minimises $\ell(P)$.



We introduce now *Dijkstra's Algorithm* (1956). For vertices x and y, this finds an ℓ-shortest xy-path.

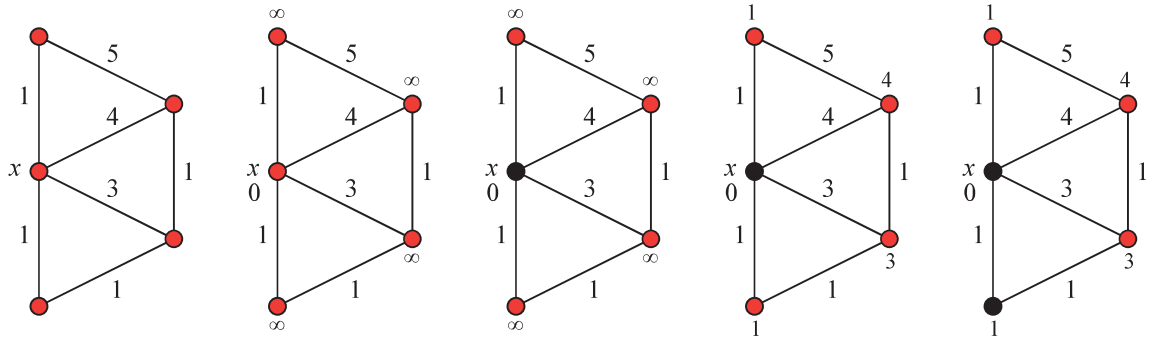
The idea of the algorithm is to maintain a 'tentative distance from x', denoted $D(v)$, for each vertex $v \in V(G)$. The $D(v)$ (non-strictly) decreases for each vertex with each step, and with each step of the algorithm we finalise $D(u)$ for one vertex u. At the end of the algorithm all $D(u)$ will be equal to the correct value, that is $D(u) = \ell(P_u^*)$ for some ℓ-shortest xu-path P_u^*.

To begin the algorithm, we set $U = V(G)$, in the context that U is the set of vertices v for which $D(v)$ has not yet been finalised – that is, *all* the vertices. We further set $D(x) = 0$ and $D(v) = \infty$ for all $v \neq x$.

We then repeat the following step:

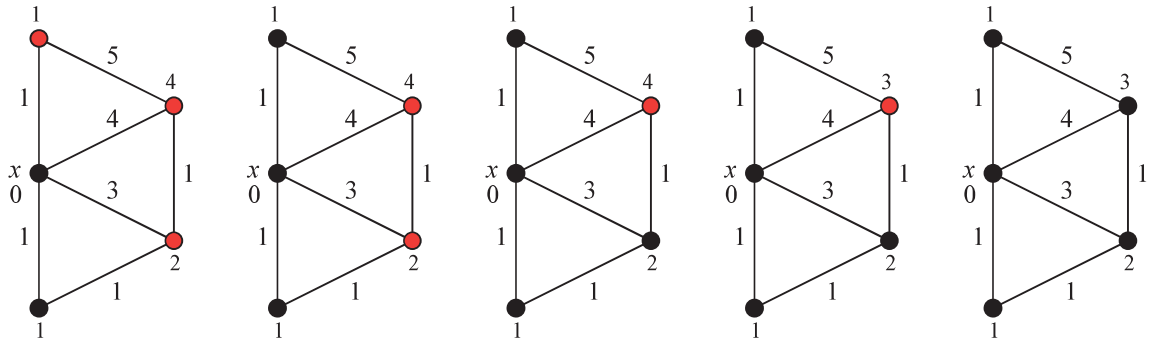
If $U = \emptyset$, the algorithm stops. Otherwise, pick $u \in U$ with $D(u)$ minimal, delete u from U, and for any $v \in U$ with v adjacent to u and satisfying $D(v) > D(u) + \ell(uv)$ replace $D(v)$ by $D(u) + \ell(uv)$.

We will label the vertices of this graph v_1, v_2, v_3 moving down the left-hand side and v_4, v_5 going down the right-hand side.

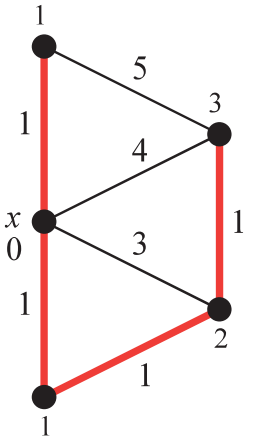


Steps above: $U = V(G)$, the red vertices; assign values of D; delete $x = v_2$ from U; all other vertices are adjacent to x and are assigned new D; delete v_3 where $D = 1$.

Steps below: Assign new values of D; delete v_1 where $D = 1$; assign new values of D (no change); delete v_5 where $D = 3$; assign new values of D; delete $y = v_4$ and now $U = \emptyset$.



Dijkstra's algorithm can be used to do more: for any $x \in V(G)$, we can construct a spanning tree T such that for any $y \in V(G)$, the unique xy-path in T is an ℓ-shortest xy-path. We call T an *ℓ-shortest path tree rooted at x*. We now describe how to obtain T. (This is the red tree for the weighted graph below.)

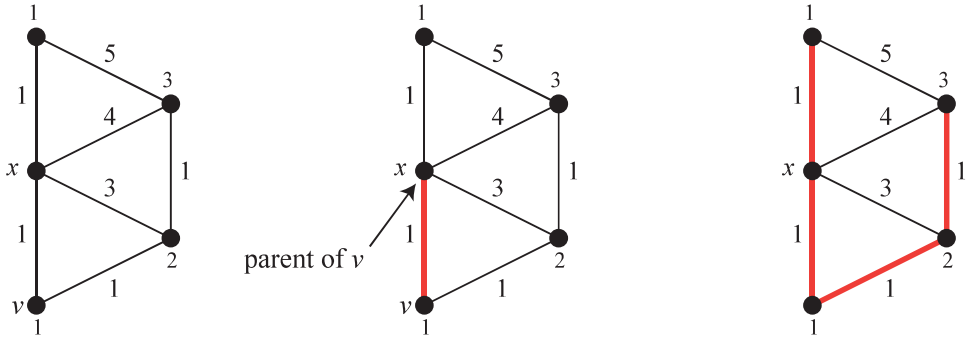


For any vertex $v \neq x$, the *parent* of v is the last vertex u such that we replaced $D(v)$ by $D(u) + \ell(uv)$ during Dijkstra's algorithm.

The steps of the above implementation are encapsulated in the table below. The values in the row of v_i cease once $D(v_i)$ has been finalized.

v_1	∞	1	1	1	
v_2	0				
v_3	∞	1			
v_4	∞	4	4	4	3
v_5	∞	3	2		

We also note that v_2 is the parent of v_3, v_3 is the parent of v_5. v_2 is the parent of v_1 and v_5 is the parent of v_4. We obtain T by drawing an edge from each vertex $v \neq x$ to the parent of v as below.



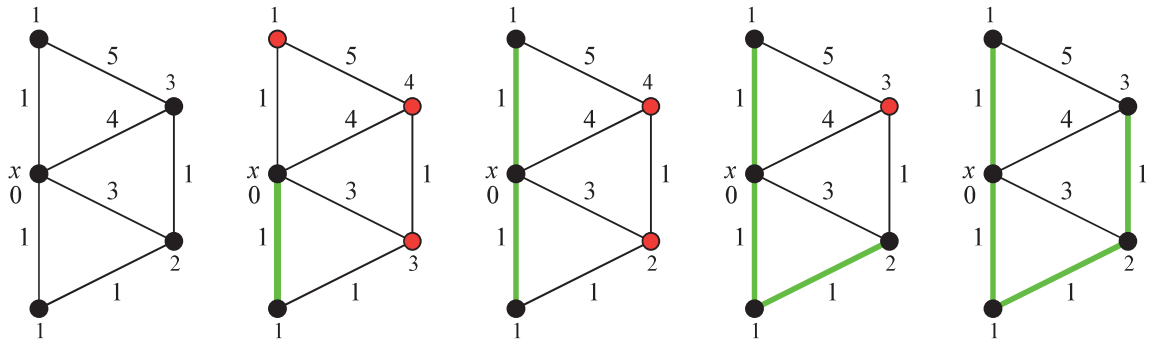
Lemma 21. *T is a tree, and for each $u \in V(G)$ we have $D(u) = \ell(P_u)$ where P_u is the unique xu-path in T.*

Proof. (Of lemma) After any step, we have defined the parents of all vertices in $C = V(G) \setminus U$. Let T_C be obtained by drawing an edge from each $v \in C \setminus \{x\}$ to its parent. So $V(T_C) = C$. We show by induction on $|C|$ that T_C is a tree and for each $u \in V(T_C)$ we have $D(u) = \ell(P_u)$ where P_u is the unique xu-path in T_C. Base case: We start with $V(T_C) = \{x\}$ and no edges, which is a tree, with $D(x) = 0 = \ell(P_x)$.

Inductive step: When we delete u from U, we add u to C, and add an edge from u to the parent v of u, i.e. we add a leaf to T_C, and so obtain another tree. By definition of parent and induction we have

$$D(u) = D(v) + \ell(vu) = \ell(P_v) + \ell(vu) = \ell(P_u).$$

□



Theorem 22. *T is an ℓ-shortest paths tree rooted at x.*

Proof. For each $u \in V(G)$, let $D^*(u) = \ell(P_u^*)$ for some ℓ-shortest xu-path P_u^*. We show by induction that in each step of the algorithm, when u is deleted we have $D(u) = D^*(u)$.

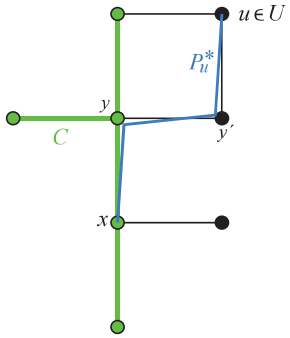
Base case: We have $u = x$ and $D(u) = D^*(u) = 0$.

Inductive step: Consider the step where we delete some u from U and suppose, for a contradiction, that $D(u) > D^*(u)$. Let $C = V(G) \setminus U$. By induction, for every vertex v in T_C, $D^*(v) = D(v)$.

Let yy' be the first edge of P_u^* with $y \notin U$ and $y' \in U$. By the inductive hypothesis $D(y) = D^*(y)$. Now

$$\begin{aligned} D(y') &\leqslant D(y) + \ell(yy') \\ &= D^*(y) + \ell(yy') \\ &= \ell(P_y^*) + \ell(yy') \\ &\leqslant \ell(P_u^*) = D^*(u) < D(u). \end{aligned}$$

The first inequality uses the update rule for y and y': when y was removed from U, either the tentative value $D(y')$ is less than $D(y)+\ell(yy')$ or it is replaced by the latter value. The second inequality holds as the subpath of P_u^* from x to y must be an ℓ-shortest xy-path or we would find a shorter path to u.



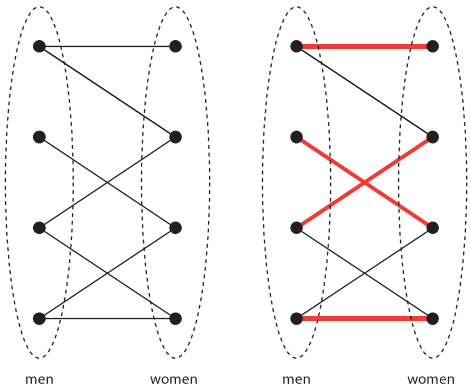
However, $y' \in U$ with $D(y') < D(u)$ contradicts the choice of u in the algorithm. So $D(u) = D^*(u)$.

□

The running time of this implementation of Dijkstra's Algorithm is $O(|V(G)||E(G)|)$. A better implementation (which we omit) gives a running time of $O(|E(G)| + |V(G)| \log |V(G)|)$.

7 Matchings

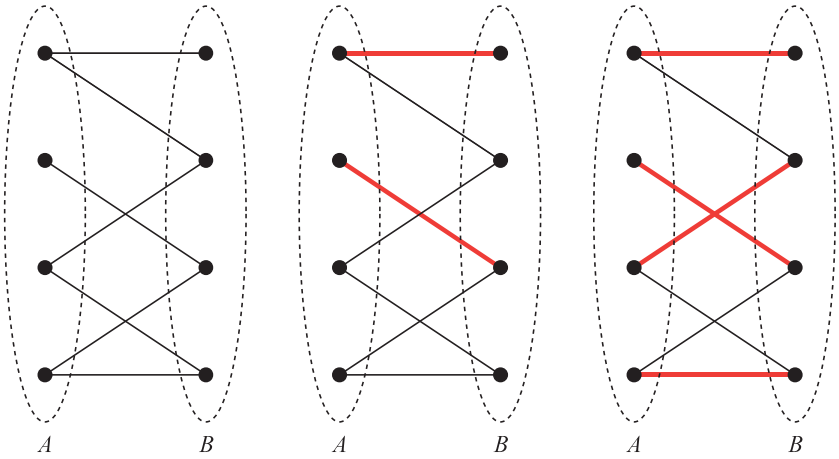
The Marriage Problem is the following: Given n men and n women, under what conditions is it possible to pair each man with a woman such that every pair knows each other?



Definition 23. *(a) A graph G is **bipartite** if we can partition $V(G)$ into two sets A and B so that every edge of G crosses between A and B.*

*(b) We say $M \subseteq E(G)$ is a **matching** if the edges in M are pairwise disjoint.*

*(c) We say a matching M is **perfect** if every vertex belongs to some edge of M.*



a bipartite graph G a matching in G a perfect matching in G.

How can we produce a matching of maximal size? The greedy algorithm does not work as shown below.

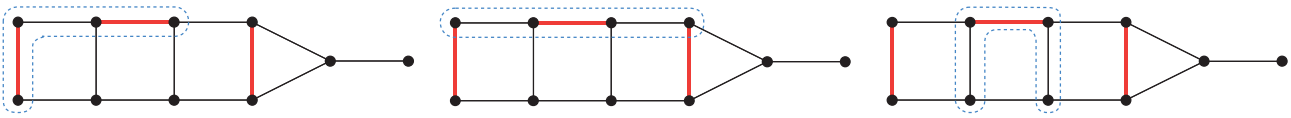
This matching can not be extended but clearly isn't maximal; a maximal matching would be the first and third edges.

We introduce first the notions of *alternating and augmenting paths.*

Definition 24. *Let G be a graph, let M be matching in G, and let P be a path in G.*

*(a) We say P is M-**alternating** if every other edge of P is in M.*

*(b) We say P is M-**augmenting** if P is M-alternating and its end vertices are not in any edge of M.*



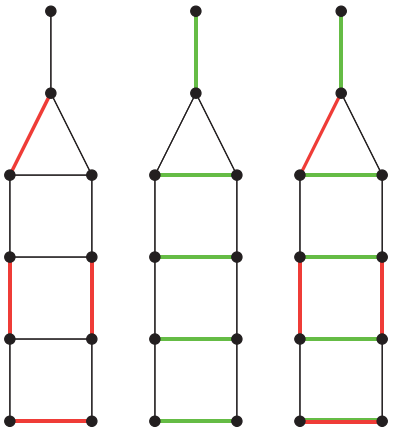
Three M-alternating paths, the last being M-augmenting

Lemma 25. *Let M be a matching in G. Then M is not of maximum size if and only if there is an M-augmenting path in G.*

Proof. If there is an M-augmenting path P in G, then we can find a larger matching by 'flipping' P: replace M by $M \setminus (M \cap E(P)) \cup (E(P) \setminus M)$. That is, those edges of P that were in M are dropped to be replaced by those that weren't.

Conversely, suppose that M^* is a matching in G with $|M^*| > |M|$. Let $H = M \cup M^*$. Every vertex has degree at most 2 in H, so each component of H is a path (which might be a single edge) or a cycle. The edges in the components alternate between M and M^*.

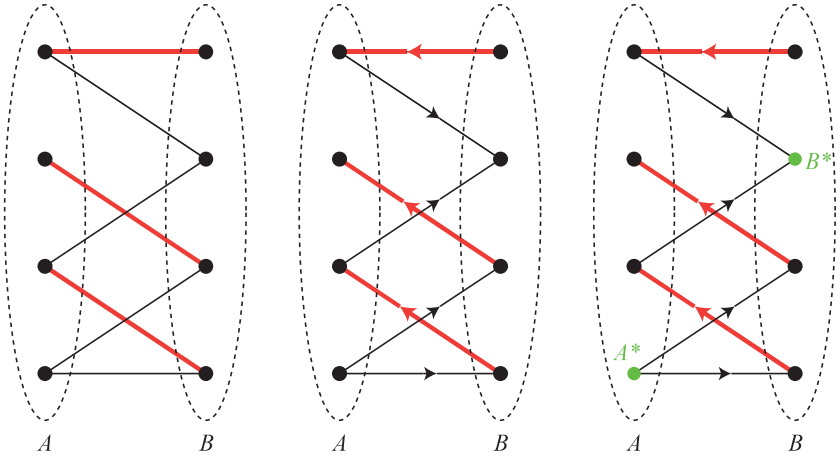
As $|M^*| > |M|$, we can find a component with more edges of M^* than M (which might just be a single edge in $M^* \setminus M$); this is an M-augmenting path in G. □



In the above graphs are sketched red M, green M^* and their union.

Lemma 25 reduces the algorithmic question of finding a maximum matching in G to the following: given a matching M in G, find an M-augmenting path or show that there is none.

We'll focus on the case of bipartite graphs. Suppose that G is bipartite, with parts A and B. Let M be a matching. We put directions on $E(G)$, so that all edges in M are one-way from B to A, and all edges not in M are one-way from A to B. Let A^* and B^* be the vertices in A and B that are 'uncovered', i.e. not in any edge of M. Then an M-augmenting path is equivalent to a directed path from A^* to B^*, i.e. a path that respects directions of edges.

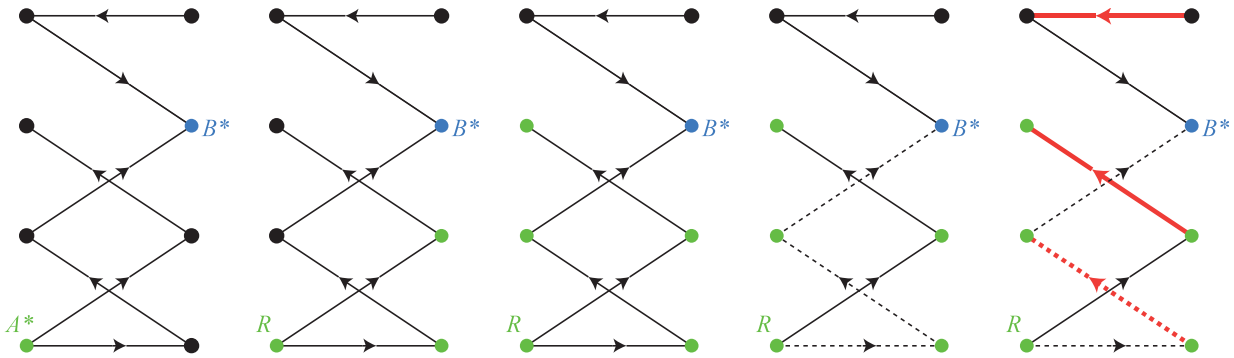


Is there a directed path from A^* to B^*? More generally, suppose that we have a directed graph with subsets A^* and B^* of $V(G)$. Is there a directed path from A^* to B^*?

Search algorithm
Start with $R = A^*$. Repeat the following step:

> if there is any edge directed from some $x \in R$ to some $y \notin R$ then add y to R, otherwise stop.

There is a directed path from A^* to B^* if and only if the final R intersects B^*.



The *Hungarian algorithm* finds a matching of maximum size in a bipartite graph G.

Start with $M = \emptyset$. Orient the edges of G: all edges in M are one-way from B to A, and all edges not in M are one-way from A to B.

Let A^* and B^* be the vertices in A and B that are 'uncovered', i.e. not in any edge of M. Use the search algorithm to find a directed path from A^* to B^*.

If there is no such path, stop. If there is, then it is M-augmenting and so we 'flip' the path to increase the size of M.

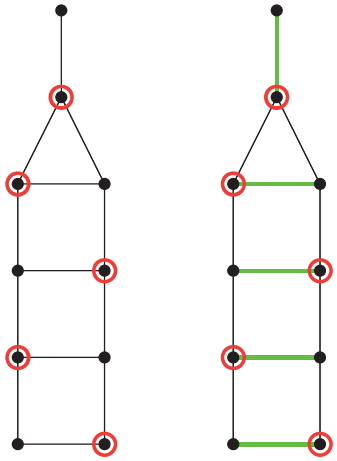
Repeat.

The running time of the search algorithm is $O(|V(G)||E(G)|)$, and there are at most $|V(G)|/2$ iterations of increasing the matching. So the Hungarian algorithm has running time $O(|V(G)|^2|E(G)|)$.

8 Matchings and covers

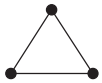
Definition 26. *A **cover** for a graph G is a subset C of the vertices such that every edge contains at least one vertex of C.*

If M is any matching and C is any cover, then $|M| \leqslant |C|$. To see this, define an injective map $f : M \to C$, where $f(e)$ is any vertex of $e \cap C$.



A matching (green edges) of equal size to a cover (red vertices)

Suppose that we had found a matching M and a cover C such that $|M| = |C|$. Then we would know that M was a maximal size matching and C was a minimal size cover. This is an example of 'weak duality' and suggests the question of whether equality holds. The answer to the question is 'no' in general:



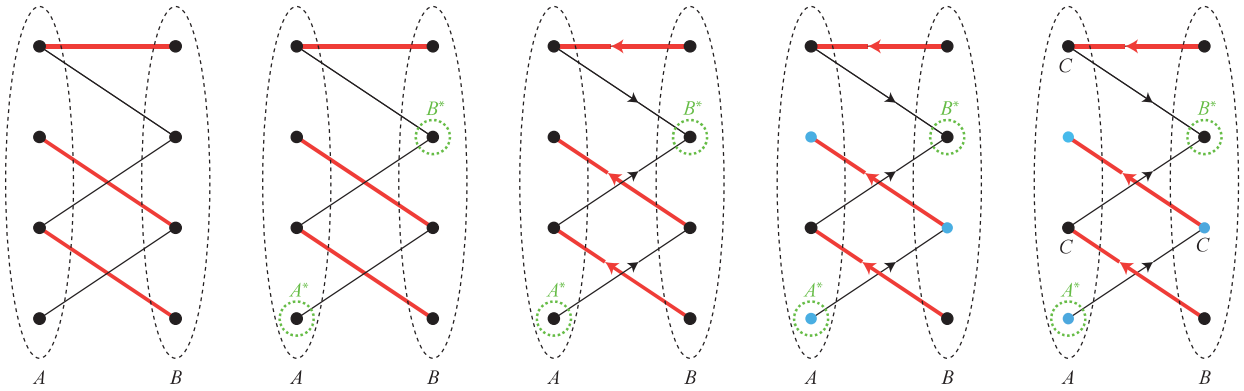
In the above graph, the maximum matching has size 1 but the minimum cover has size 2.

Theorem 27. ***(König's Theorem (1931))*** *In any bipartite graph, the size of a maximum matching equals the size of a minimum cover.*

Proof. Let G be a bipartite graph, with parts A and B, and let M be a maximum matching in G. It suffices to find a cover C with $|C| = |M|$.

Recall that we write A^* and B^* for the uncovered vertices in A and B. Consider the search algorithm for an M-augmenting path in G. The algorithm terminates with some set R that consists of all vertices reachable by M-alternating paths starting in A^*. As M is maximum there is no M-augmenting path, so $R \cap B^* = \emptyset$.

Let $C = (A \setminus R) \cup (B \cap R)$. We claim that C is a cover with $|C| = |M|$.



We start by showing that C is a cover. Suppose not. Then there is $ab \in E(G)$ with $a \in A \cap R$ and $b \in B \setminus R$. However, this contradicts the definition of R: if $ab \notin M$ we can reach b via a and if $ab \in M$ we cannot reach a via b. Thus C is a cover.

It remains to show $|C| = |M|$. It suffices to show that every vertex in C is covered by some edge of M, and that no edge of M covers two vertices of C. (This will show $|C| \leqslant |M|$, and we noted previously that $|M| \leqslant |C|$ is immediate from the definitions.)

Recall that $C = (A \setminus R) \cup (B \cap R)$. If $a \in A \setminus R$, then a is covered by M as $A^* \subseteq R$. And if $b \in B \cap R$, then b is covered by M, or else $b \in B^* \cap R = \emptyset$ which gives a contradiction; the set is empty as M is maximal and so there is no M-augmenting path.

Finally suppose for a contradiction that an edge ab with both ends in C is covered by M. That is $ab \in M$ with $a \in A \setminus R$, $b \in B \cap R$. We can thenreach a via b, contradicting $a \notin R$.

We conclude $|C| = |M|$.

□

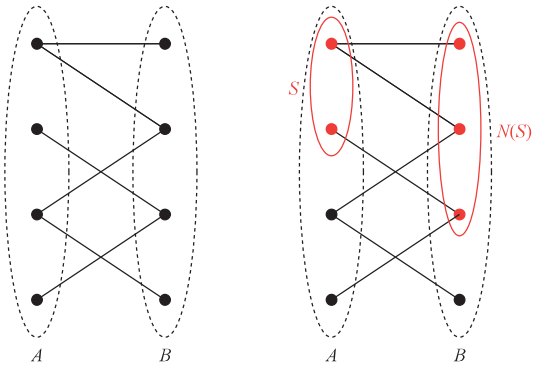
The marriage problem: Let G be a bipartite graph with parts A and B. We consider the more general question of whether there is a matching that covers every vertex in A; if $|B| = |A|$ then this will be perfect.

For $S \subseteq A$ the *neighbourhood* of S is

$$N(S) = \bigcup_{a \in S} \{b \mid ab \in E(G)\}.$$

Note that if G has a matching M covering A then each $a \in S$ has a 'match' a' with $aa' \in M$, and the matches are distinct, so $|N(S)| \geqslant |S|$.

This gives a necessary condition for G to have a matching; it is also sufficient ...



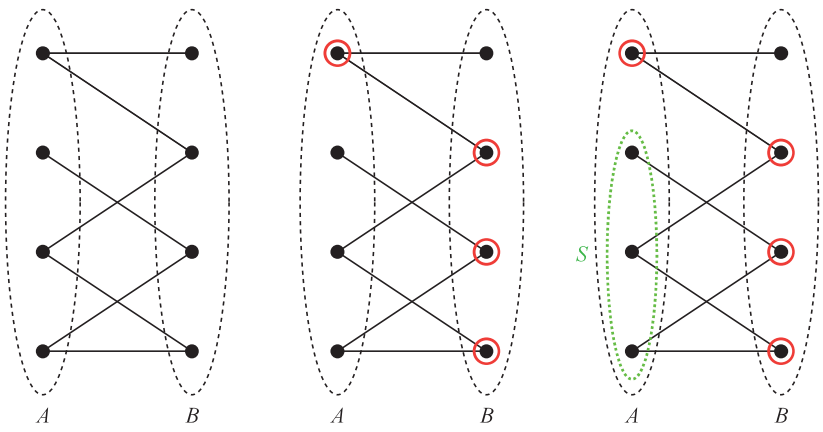
Theorem 28. ***(Hall's Theorem (1935))*** *Let G be a bipartite graph with parts A and B. Then G has a matching covering A if and only if every $S \subseteq A$ satisfies $|N(S)| \geqslant |S|$.*

Proof. We have already remarked that the above condition is necessary. Conversely, suppose that every $S \subseteq A$ satisfies $|N(S)| \geqslant |S|$. Let C be any cover of G. By König's Theorem, it suffices to show $|C| \geqslant |A|$.

Let $S = A \setminus C$. Note that by definition of a cover, we have $N(S) \subseteq B \cap C$. Then

$$|C| = |A \cap C| + |B \cap C| \geqslant |A| - |S| + |N(S)| \geqslant |A|.$$

□



9 The Chinese Postman Problem

A postman collects a sack of letters from the sorting office, walks along every street to deliver them, and returns to the office. How can they find the shortest route? (The problem was first posed by the Chinese mathematician Kwan Mei-Ko in 1960 and is named in his honour.)

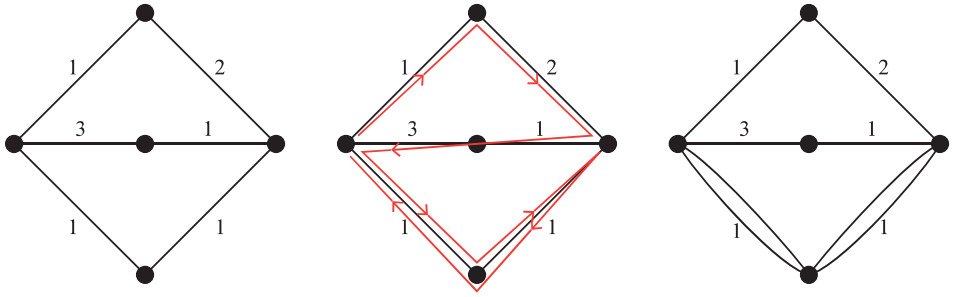
Let G be a connected graph, and let W be a closed walk in G. We call W a *postman walk* in G if it uses every edge of G at least once.

For each $e \in E(G)$, let $c(e) > 0$ be the length of e. The length of W is $c(W) = \sum_{e \in W} c(e)$.

We want to find a shortest postman walk.

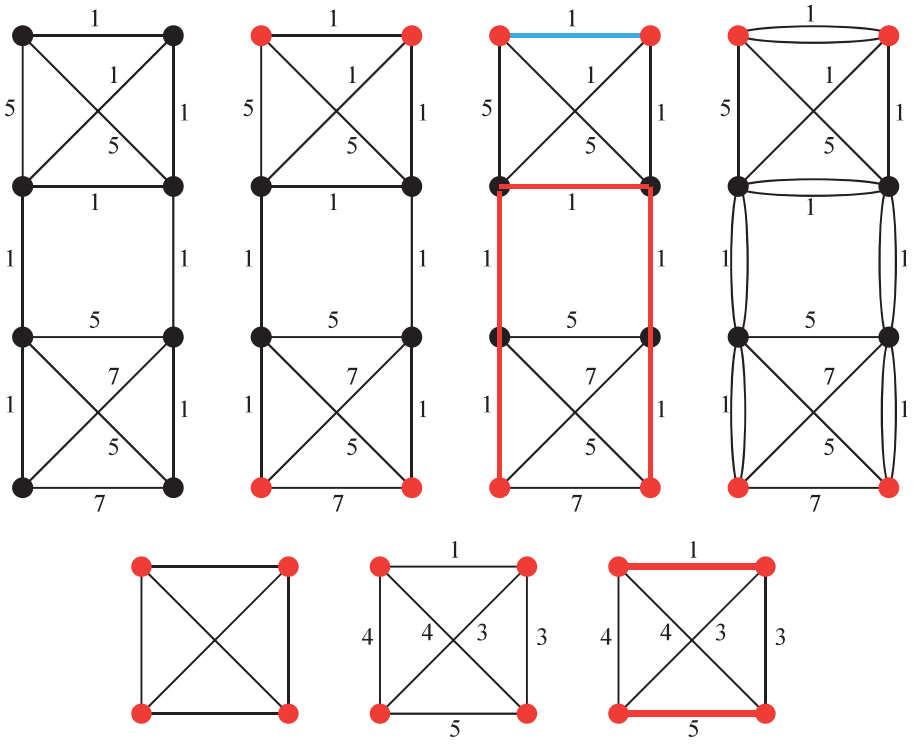
We can interpret a postman walk W as an Euler Tour in an *extension* of G, in which we introduce parallel edges, so that the number of parallel edges joining vertices x and y is the number of times that xy is used in W.

Thus an equivalent reformulation of the Chinese Postman Problem is to find a *minimum weight Eulerian extension* G^* of G, i.e. G^* is obtained from G by copying some edges, so that all degrees in G^* are even, and $c(G^*)$ is as small as possible. Note that such an extension is not a simple graph, but rather is a multigraph.



We can see in the above case that we have a minimum weight Eulerian extension; there are two vertices with odd degrees, so at least two edges need adding into the extension and in each case those edges have minimal cost.

We now describe *Edmonds' algorithm* (1973). We *assume* that we have access to an algorithm for finding a minimum weight, perfect matching in a weighted graph. (An algorithm for this problem was also found by Edmonds, but it is beyond the scope of this course).



1. Let X be the set of vertices with odd degree in G. (These are coloured red in the second graph of the first row.)
2. For each $x \in X$, find a c-shortest path tree T_x rooted at x. (We have previously shown that Dijkstra's algorithm yields such a tree.)
3. Define a weight function w on pairs in X: let

$$w(xy) = c(P_{xy}),$$

   where P_{xy} is the unique xy-path in T_x. (This weight function is calculated in the second graph on the second row.)
4. Find a perfect matching M on X with minimum w-weight. Note that this perfect matching step makes sense as $|X|$ is even, by the handshaking lemma. (This matching is drawn in the third graph on the second row. The associated paths are drawn in the third graph of the first row.)
5. Let G^* be the Eulerian extension of G obtained by copying all edges of P_{xy} for all $xy \in M$. (The extension by these two paths appears in the fourth graph of the first row.)
6. Find an Euler Tour W in G^*. (Fleury's algorithm yields such a tour.)
7. Interpret W as a postman walk in G.

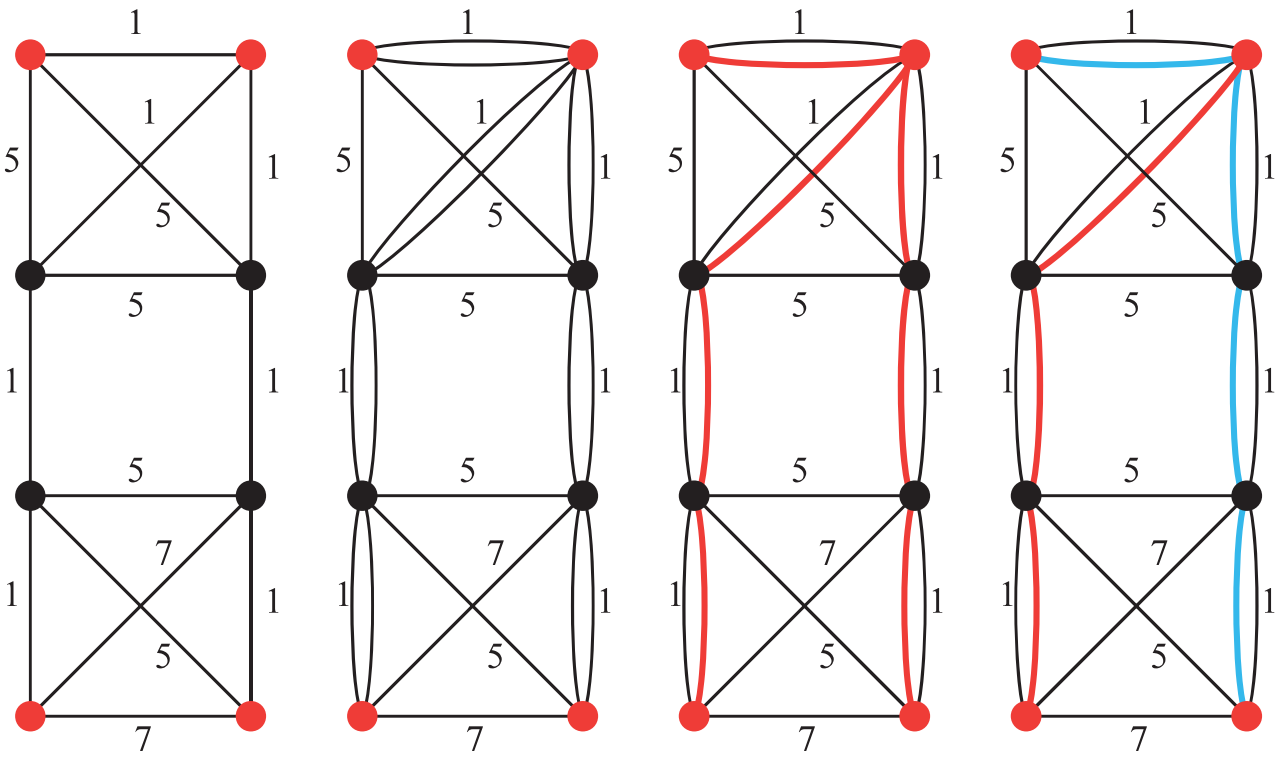
Lemma 29. *Let H be a graph in which not all degrees are even. Then there is a path in H such that both ends have odd degree.*

Proof. Pick a component of H containing a vertex of odd degree. By the handshaking, there is another vertex of odd degree in H. Pick a path joining these two vertices. □

Theorem 30. *Edmonds' Algorithm finds a minimum length postman walk.*

Proof. Let W^* be a minimum length postman walk. (Such is depicted in the second diagram below.) It suffices to show that Edmonds' algorithm finds a postman walk that is no longer than W^*.

Let G^* be the Eulerian extension of G defined by W^*. Let H be the graph of repeated edges: $E(H) = E(G^*) \setminus E(G)$. (These are depicted in the third diagram below.) Note that the set of vertices with odd degree in H is X (i.e. the same set as for G).



We construct a set of paths in H by repeating the following procedure: if the current graph has any vertices of odd degree, apply Lemma 29 to find a path P such that both ends have odd degree, delete the edges of P and repeat. (Such a blue path P is drawn in the fourth diagram below.) This procedure pairs up the vertices in X, so that each pair is connected by a path in H.

Let $H' \subseteq H$ be the graph formed by the union of these paths. Let G' be the Eulerian extension of G defined by copying the edges of H'. Let W' be an Euler tour in G', interpreted as a postman walk in G. Then $c(W') \leqslant c(W^*)$.

By definition of the algorithm, it finds a postman walk that is no longer than W'.

□