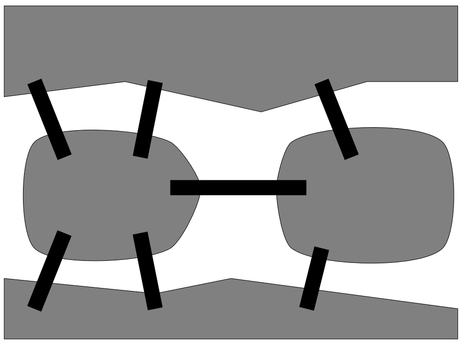
Chapter 6

Traversability

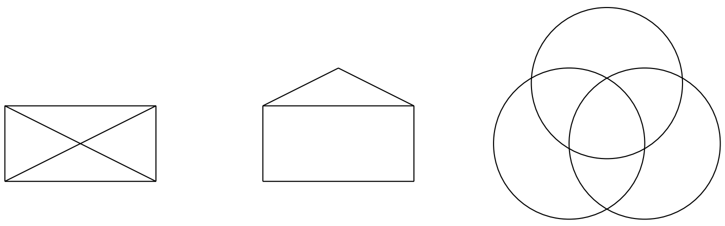
In Chapter 5 we studied (connected) graphs that contain no cycles, and we saw that for any vertices v and w in such a graph there is precisely one route from v to w. In this chapter we will consider graphs for which there may be multiple possible routes, and we will consider problems relating to finding routes that are 'best' possible, according to some criterion. We will see that cycles play an important role in these sorts of problems. (Recall that a trail is a walk with no repeated edges, and a cycle is a closed trail in which none of the intermediate vertices are repeated.) We begin with some classical problems to motivate the sorts of questions we will study.

Exercise 6.1 (The Seven Bridges of Königsberg)**.** The town of Königsberg was situated by the River Pregel, and had seven bridges connecting two islands in the river to each other and to the mainland.



A popular excursion for the townsfolk was to stroll across the bridges in an attempt to find a route that would take them across each of the bridges exactly once. Can you find such a route?

Exercise 6.2. Which of these diagrams can be drawn without taking your pen off the paper or retracing a line?



Exercise 6.3. A *domino* is a rectangle divided into two squares that each contain a number between 0 and n. A *set of dominoes* consists of a collection of each of the $\binom{n+1}{2}$ possible dominoes (repetition of numbers within a domino is allowed, but order does not matter, so $\boxed{3}\boxed{4}$ is the same as $\boxed{4}\boxed{3}$.) When playing the game of dominoes, two dominoes can be placed end-to-end if the corresponding numbers agree: $\boxed{3}\boxed{4}\ \boxed{4}\boxed{1}\ \boxed{1}\boxed{0}$. Is it possible to arrange an entire set of dominoes into a circle so that all adjacent numbers match in this fashion?
e.g. consider $n = 4$. In this case the set of dominoes is:

$$\begin{array}{ccccc}
\boxed{2}\boxed{3} & \boxed{2}\boxed{4} & \boxed{3}\boxed{3} & \boxed{3}\boxed{4} & \boxed{4}\boxed{4} \\
\boxed{1}\boxed{1} & \boxed{1}\boxed{2} & \boxed{1}\boxed{3} & \boxed{1}\boxed{4} & \boxed{2}\boxed{2} \\
\boxed{0}\boxed{0} & \boxed{0}\boxed{1} & \boxed{0}\boxed{2} & \boxed{0}\boxed{3} & \boxed{0}\boxed{4}.
\end{array}$$

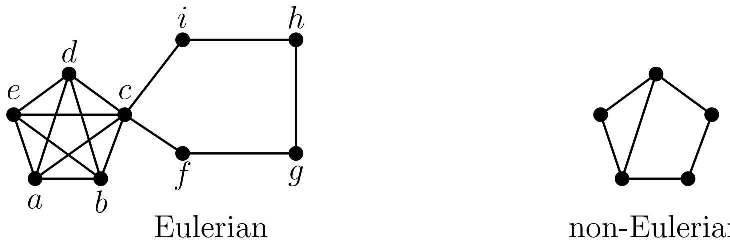
What about the case $n = 3$?

6.1 Eulerian Graphs

The Königsberg bridge problem was studied by the Swiss mathematician Leonhard Euler, and is considered to be one of the earliest problems in graph ever studied.

Definition 6.1. A graph G is said to be *Eulerian* if it contains a closed trail that includes every edge. Such a trail is called an *Eulerian trail.*

Example 6.4.



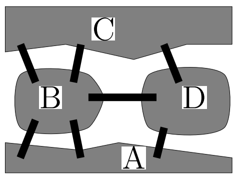
Eulerian non-Eulerian

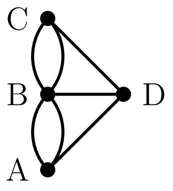
Eulerian trail: $c, f, g, h, i, c, d, e, a, b, c, e, b, d, a, c$

We note that for a graph to be Eulerian, it must be connected. If a graph G has an Eulerian trail, then there is an Eulerian trail starting at any vertex of G.

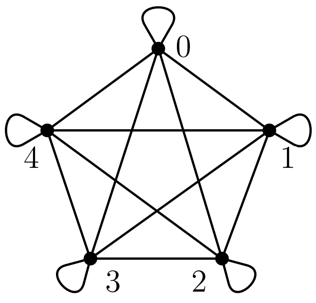
Each of the exercises at the start of this chapter can be expressed in terms of finding an Eulerian trail in a graph.

The bridges of Königsberg





Dominoes We can represent a set of dominoes by a graph with vertices labelled with the integers $0, 1, 2, \ldots, n$, in which an edge joining vertex i to vertex j represents the domino $\boxed{i|j}$.



An example of an Eulerian trail in this graph is $3, 4, 4, 0, 0, 1, 2, 2, 4, 1, 1, 3, 0, 2, 3, 3$ (check it for yourself!)

How can we tell if a graph is Eulerian? Fortunately there is a straightforward characterisation of Eulerian graphs that allows us to answer this question.

Theorem 6.1. *A connected graph G is Eulerian if and only if each vertex has even degree.*

Proof. $\Rightarrow$ If G is Eulerian then it has a closed trail containing each edge. As the trail is followed around the graph, each time a given vertex v is included in the trail there are two corresponding edges incident with v that belong to the trail, hence the degree of v must be even for any vertex v of G.

$\Leftarrow$ Now suppose that G is a connected graph in which the degree of each vertex is even. We use strong induction on the number of edges to show that G is Eulerian. Suppose that for some integer k, every connected graph with k or fewer edges whose vertex degrees are all even is Eulerian. Now suppose G is a connected graph with $k + 1$ edges whose vertex degrees are all even. If the number of vertices in G is n, then the sum of their degrees is at least $2n$, so by the handshaking lemma the number of edges is at least n, and hence G contains a cycle. We denote the cycle by C. Let H be the graph obtained by removing the edges of C from G. In doing this, the degree of any vertex that belongs to C is reduced by 2, and the degrees of all other vertices are unchanged. This implies that the degree of each vertex in H is even, although it is not necessarily connected. By the inductive assumption, each connected component of H has an Eulerian trail. We can then construct an Eulerian trail for G in the following manner:

1. Begin with any vertex in C. Follow the edges of C until a vertex is reached that belongs to a component of H that has more than one vertex.

2. Follow the edges of an Eulerian trail in that component of H.
3. Continue following the edges of C; each time a vertex belonging to a component of H with more than one vertex is reached add in the edges of an Eulerian trail in that component. This process is repeated until the initial vertex is reached.

□

Exercise 6.5.

1. Which of the complete graphs are Eulerian?
2. Which of the complete bipartite graphs are Eulerian?
3. Which of the Platonic graphs are Eulerian?

The proof of Theorem 6.1 can be adapted to prove the following corollary:

Corollary 6.2. *A connected graph G is Eulerian if and only if its edges can be partitioned into edge-disjoint cycles.*

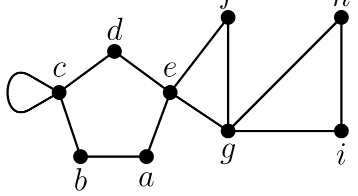
Now that we have a means to determine whether a graph is Eulerian, we would like to know how to find an Eulerian trail in an Eulerian graph. This can be accomplished by a method known as *Fleury's algorithm.*

Theorem 6.3 (Fleury's algorithm)**.** *The following algorithm takes an Eulerian graph G as input, and outputs an Eulerian trail in G.*

1. *Start with any vertex of G.*
2. *Add an edge to the end of the trail. This edge can be chosen arbitrarily, subject to the condition that an edge which is a* bridge[1] *is only chosen if there is no other option.*
3. *Delete this edge that was just chosen from G, and delete any resulting isolated vertices.*
4. *Repeat the previous two steps until the trail is complete.*

It can be shown that this process always gives an Eulerian trail when applied to an Eulerian graph.

Example 6.6. Use Fleury's algorithm to find an Eulerian trail in the following graph:



1 Start at g.

2 Go to h;

[1]A bridge is an edge whose removal causes the graph to be disconnected.

3 delete gh.

2 Go to i;

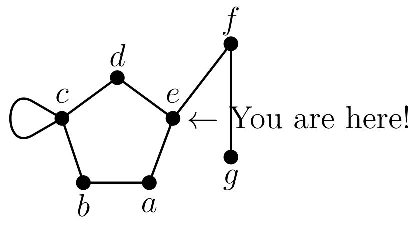
3 delete hi and h.

2 Go to g;

3 delete ig and i.

2 Go to e;

3 delete ge.



2 Cannot go to f as ef is a bridge. Go to a;

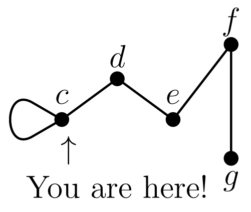
3 delete ea.

2 Go to b;

3 delete ab and a.

2 Go to c;

3 delete bc and b



2 Cannot go to d as cd is a bridge. Go to c;

3 delete cc.

2 Go to d;

3 delete cd and c.

2 Go to e;

3 delete de and d.

2 Go to f;

3 delete ef and e.

2 Go to g;

3 delete fg and f and g. STOP!

Eulerian trail: $g, h, i, g, e, a, b, c, c, d, e, f, g$

Exercise 6.7. Use Fleury's algorithm to find an Eulerian trail in the octahedron graph.

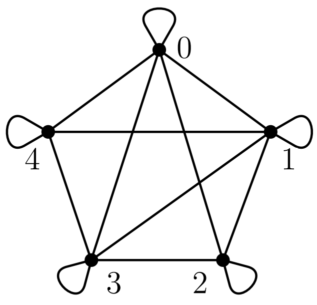
Semi-Eulerian Graphs

We say that a graph G is *semi-Eulerian* if G is not Eulerian, but there is a trail that contains every edge of G. Such a trail is known as a semi-Eulerian trail. Semi-Eulerian graphs can be characterised by the following corollary to Theorem 6.1.

Theorem 6.4. *A connected graph G is semi-Eulerian if and only if it has precisely two vertices of odd degree.*

Any semi-Eulerian trail in a semi-Eulerian graph G starts and ends at the vertices of odd degree. Fleury's algorithm can be adapted to find a semi-Eulerian trail simply by specifying that the first vertex chosen must have odd degree.

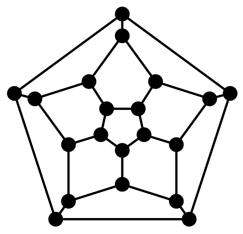
Exercise 6.8. Use Fleury's algorithm to find a semi-Eulerian trail in the following graph:



6.2 Hamiltonian Cycles

We have considered the problem of finding trails that pass through every edge of a graph once. Now we consider the problem of finding cycles that visit every vertex precisely once.

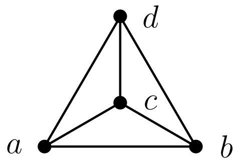
Exercise 6.9 (Hamilton's Icosian Game)**.** The following puzzle was proposed by the Irish mathematician William Hamilton:



Find a cycle in the above graph that visits every vertex precisely once.

Definition 6.2. A graph G is said to be *Hamiltonian* if there exists a cycle that contains all the vertices of G. Such a cycle is known as a *Hamiltonian cycle*.

Example 6.10.



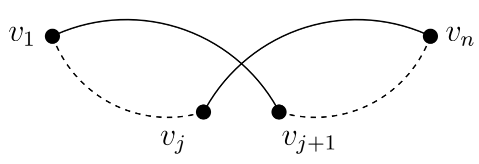
An example of a Hamiltonian cycle in the above graph is a, b, c, d, a.

At first glance the concept of a Hamiltonian cycle seems very similar to that of an Eulerian trail. However, in practice they turn out to be somewhat trickier to work with. We have seen that the problem of deciding whether a graph is Eulerian can be solved by simply considering the degrees of the vertices. However, we have no such simple characterisation in the case of Hamiltonian graphs. Indeed, the problem of determining in general whether a given graph is Hamiltonian is believed to be computationally difficult. About the best we can say is that if a graph has "enough" edges then it is necessarily Hamiltonian. We will see two different theorems, due to Ore and Dirac, that give sufficient conditions for a graph to be Hamiltonian. Without loss of generality we will restrict our attention to simple graphs, since a loop can never be part of a Hamiltonian cycle, and if vertices v and w are joined by multiple edges, then at most one of them will be part of any Hamilton cycle.

Theorem 6.5 (Ore's Theorem). *Let G be a connected simple graph with $n \geq 3$ vertices. If $d(u) + d(v) \geq n$ for any non-adjacent pair of vertices u and v, then G is Hamiltonian.*

Proof. Let G be a connected simple graph with $n \geq 3$ vertices that satisfies $d(u) + d(v) \geq n$ for any non-adjacent pair of vertices u and v, and suppose G is not Hamiltonian. Without loss of generality we can assume that G has the property that the addition of any additional edge would create a Hamiltonian cycle. (For, if it did not, we could simply add extra edges until this property was fulfilled, and it would still be the case that the sum of the degrees of any pair of non-adjacent vertices would be at least n.)

Thus there must exist a spanning path $P = v_1, v_2, \ldots, v_{n-2}, v_n$. Let S be the set of all vertices adjacent to v_1, and let R be the set of all vertices v_i that occur in P immediately prior to some vertex v_{i+1} that lies in S. Then $|R| = |S| = d(v_1)$. Since $d(v_1) + d(v_n) \geq n$, it follows that at least one vertex $v_j \in R$ is adjacent to v_n. But then $v_1, v_2, \ldots, v_j, v_n, v_{n-1}, v_{n-2}, \ldots, v_{j+1}, v_1$ is a Hamiltonian cycle in G, which contradicts our initial assumption.

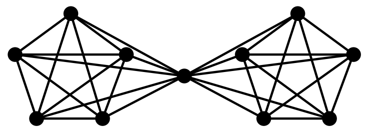


□

Example 6.11. The graph $K_{3,3}$ has six vertices and is regular of degree 3. Thus the sum of the degrees of any two non-adjacent vertices is $3 + 3 = 6 = n$, and so it must be Hamiltonian.

However, we note that the converse of Ore's theorem is not true. For example, C_5 is Hamiltonian, yet the sum of the degrees of any two non-adjacent vertices is $2 + 2 = 4 < 5$.

We further observe that the condition $d(v) + d(u) \leq n$ is tight: the result is not true if n is replaced by $n - 1$, as illustrated by the following graph, which consists of two copies of K_5 plus one extra vertex that is joined to all the other vertices.



Here any two vertices that are non-adjacent each have degree 5, so the sum of their degrees is $5+5 = 10 = n-1$, and yet the graph is not Hamiltonian. (This construction can be generalised to two copies of K_n for any n.)

Corollary 6.6 (Dirac's Theorem). *Let G be a simple connected graph with $n \geq 3$ vertices. If $d(v) \geq n/2$ for all vertices v, then G is Hamiltonian.*

This result follows immediately from Ore's theorem, since any simple connected graph G for which all vertices have degree at least $n/2$ certainly has the property that the sum of the degrees of any non-adjacent vertices is at least n.

Exercise 6.12.

1. Which of the complete graphs are Hamiltonian?

2. Which of the complete bipartite graphs are Hamiltonian? *(harder)*

occurs as bits 4 and 1 (considering the sequence cyclicly) and the

6.3 The Travelling Salesman Problem

We now consider a much-studied optimisation problem.

Exercise 6.13. A travelling salesman from town a is going on a trip to try and sell his product in towns b, c, d, e, and f. If the distances between the towns in miles are as given in the following table, what is the shortest distance he will have to drive in order to visit all five towns and return home (assuming he visits no town more than once)?

	b	c	d	e	f
a	623	440	210	60	350
b	–	125	200	200	123
c	–	–	130	50	230
d	–	–	–	100	75
e	–	–	–	–	110

.

Exercise 6.14. A traveller wants to visit 5 countries A, B, C, D and E, starting and ending at her home country H. The minimum cost, in pounds, of a flight between each of the countries is given in the following table:

	A	B	C	D	E
H	250	200	300	100	150
A	–	200	250	150	100
B	–	–	100	150	150
C	–	–	–	300	350
D	–	–	–	–	200

Assuming she can visit the countries in any order and only wants to visit each of them once, what is the cheapest possible tour?

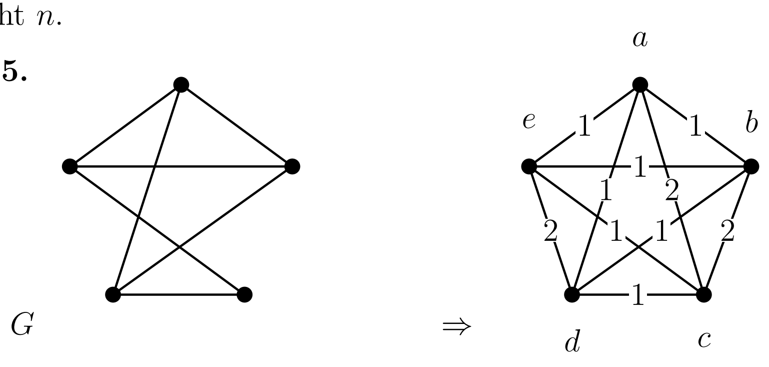
These problems are instances of the *Travelling Salesman Problem (TSP)*. In graph theoretic terms, the TSP can be expressed as follows:

Travelling Salesman Problem: Given a weighted complete graph, find a Hamiltonian cycle of minimum total weight.

The travelling salesman problem is at least as hard as the problem of determining whether a simple graph is Hamiltonian:

Given a simple graph G on n vertices, add new edges until the complete graph K_n is obtained. Assign the weight 1 to all the edges of G, and the weight 2 to all the new edges. If we are able to solve the TSP on this weighted graph, then we can determine whether G is Hamiltonian since, by construction, G is Hamiltonian if and only if the solution to the TSP has total weight n.

Example 6.15.



a, b, d, c, e, a has total weight 5, therefore this cycle is a Hamiltonian cycle in G.

The TSP is believed to be hard in general. To obtain an idea of why this is the case, consider the fact that K_n contains $\frac{1}{2}(n-1)!$ distinct Hamiltonian cycles (exercise: why is this the case?) Suppose it takes a computer 10^{-6} seconds to find the total weight of a Hamiltonian cycle. Then if we search for a solution to the TSP by finding the total weight of each Hamiltonian cycle, and choosing one with minimum weight, for particular values of n it will take the following length of time:

n	5	10	15	20	25
time	1.2×10^{-5} s	0.18144 s	$\approx$ 12.1 hours	$\approx$ 1900years	$\approx 10^{10}$years

Although it may be hard to find the best possible solution, we may still want to obtain some idea of the approximate weight of the best possible solution. Here we consider algorithms that can be used to obtain upper and lower bounds for the weight of the solution of the TSP.

An Upper Bound Algorithm for the TSP

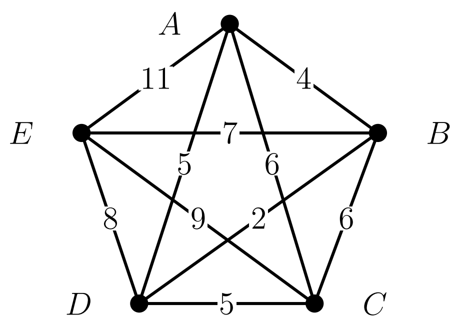
1. Choose any vertex, and find an edge of minimum weight incident with that vertex. Draw the vertices incident with the edge and join them with two edges to form a cycle.

2. Find an edge of minimum weight joining a vertex v in the cycle to a vertex w not in the cycle. Insert w into the cycle, after v in a clockwise direction.

3. Repeat the previous step until all the vertices are included in the cycle.

The result of this algorithm is a Hamiltonian cycle, and hence the weight of the edges in this cycle give an upper bound for the weight of the solution to the TSP.

Example 6.16. Use the Upper Bound Algorithm to find an upper bound for the solution to the TSP given by the following weighted graph.



1. Choose vertex B. The adjacent edge with minimum weight is BD, of weight 2. Form the cycle B, D, B.

2. An edge of minimum weight joining a vertex not in the cycle to one that is is AB. Add A to the cycle after B in a clockwise direction, to obtain B, A, D, B.

3. An edge of minimum weight joining a vertex not in the cycle to one that is is CD. Add C to the cycle after D in a clockwise direction, to obtain B, A, D, C, B.

4. An edge of minimum weight joining a vertex not in the cycle to one that is is EB. Add E to the cycle after B in a clockwise direction, to obtain B, E, A, D, C, B.

At this point all the vertices have been used. The weight of the cycle thus obtained is $5 + 5 + 6 + 7 + 11 = 34$. Thus any solution to the TSP has total weight ≤ 34.

This algorithm is fast, and so we could repeat it starting with each different vertex and take the minimum value obtained as an upper bound. Any Hamiltonian cycle can be used to bound the solution to the TSP, but this algorithm will usually give a better bound than a cycle chosen at random.

A Lower Bound Algorithm for the TSP

1. Choose any vertex v and find the minimum weights w_1 and w_2 of two edges incident with v.

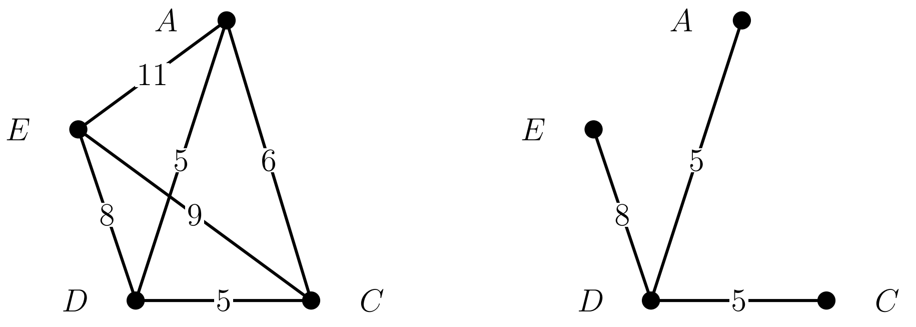
2. Delete v (and all edges incident with v) from the graph. Find a minimum spanning tree T of the resulting graph. Let W be the total weight of T.

Then a solution to the TSP has weight $\geq W + w_1 + w_2$.

Example 6.17. Use the Lower Bound Algorithm to find a lower bound to the TSP for the graph given in Example 6.16.

1. Choose B. Then w_1 and w_2 are 2, 4 respectively.

2. Delete B.



$W = 5 + 5 + 8 = 13$

Thus any solution to the TSP has total weight $\geq 18 + 2 + 4 = 24$.

As for the upper bound algorithm, we can repeat the lower bound algorithm for each vertex and choose the maximum value obtained as our bound. We can use Prim's algorithm or Kruskal's algorithm to find the minimum spanning tree.

To see why this algorithm works, suppose the cycle $x_1, x_2, \ldots, x_n, x_1$ is a solution to the TSP. Then the sum of the weights of x_1x_2 and x_nx_1 is $\geq w_1 + w_2$. Furthermore, $x_2, \ldots, x_n$ is a spanning tree of the graph with x_1 deleted. Thus the total weight of $x_2, \ldots, x_n$ is $\geq W$. Hence the total weight of $x_1, x_2, m \ldots, x_n, x_1 \geq W + w_1 + w_2$.

Exercise 6.18. Use the Upper Bound Algorithm and the Lower Bound Algorithm to find bounds for the solution of the instances of the TSP described in Exercises 6.13 and 6.14.

Learning Outcomes

After completing this chapter and the related problems you should be able to:

- Recall the definitions of a trail and a cycle;

- Understand what it means for a graph to be Eulerian;

- State necessary and sufficient conditions for a graph to be Eulerian or semi-Eulerian, and apply these conditions to determine whether or not a given graph is Eulerian or semi-Eulerian;
- Apply Fleury's algorithm to find an Eulerian trail or semi-Eulerian trail in a graph;
- Appreciate that the edges of an Eulerian graph can be partitioned into edge-disjoint cycles;
- Understand what it means for a graph to be Hamiltonian;
- State and apply Ore's theorem and Dirac's theorem;
- Appreciate that Ore's theorem and Dirac's theorem give sufficient conditions but not necessary conditions for a graph to be Hamiltonian;
- State the general form of the travelling salesman problem, and discuss the fact that it is believed to be difficult in general;
- Use the Upper Bound Algorithm and Lower Bound Algorithm to find bounds for the weight of a solution to the travelling salesman problem on a given weighted graph.