2.6.3 Find $\kappa(G)$ for the graphs G of Figure 2.32. If $\kappa(G) = 1$, identify the cut vertices of G.

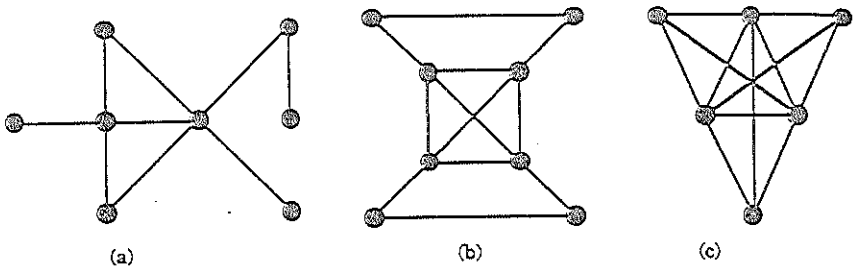


Figure 2.32

2.6.4 Give an example of a simple connected graph G with n vertices having a cut vertex v such that $\omega(G - v) = n - 1$ and each connected component of $G - v$ consists of an isolated vertex.

2.6.5 Let T be a tree with at least three vertices. Prove that there is a cut vertex v of T such that every vertex adjacent to v, except for possibly one, has degree 1.

2.6.6 Let v be a cut vertex of the simple connected graph G. Prove that v is *not* a cut vertex of its complement $\overline{G}$. (Hint: see Exercise 1.6.12.)

2.6.7 Let G be a simple connected graph with at least two vertices and let v be a vertex in G of smallest possible degree, say k.

(a) Prove that $\kappa(G) \leq k$.

(b) Prove that $\kappa(G) \leq 2e/n$, where e is the number of edges and n is the number of vertices in G.

2.6.8 Let $v_1, v_2, \ldots, v_n$ be n distinct vertices of the n-connected graph G and form the supergraph H of G by introduction of a new vertex v, not in G, which is adjacent to each of $v_1, v_2, \ldots, v_n$. Prove that H is n-connected.

2.6.9 Let G be an n-connected graph and let H be the join $G + K_1$. Prove that H is $(n+1)$-connected.

Chapter 3

Euler Tours and Hamiltonian Cycles

3.1 Euler Tours

Recall, from Section 1.6, that a trail in a graph G is a walk in G in which the edges are distinct, i.e., no edge of G appears in the trail more than once.

A trail in G is called an **Euler trail** if it includes every edge of G.

Thus a trail is Euler if each edge of G is in the trail exactly once.

A **tour** of G is a closed walk of G which includes every edge of G at least once.
An **Euler tour** of G is a tour which includes each edge of G *exactly* once.

Thus an Euler tour is just a closed Euler trail.

A graph G is called **Eulerian** or **Euler** if it has an Euler tour.

For example, the graphs G_1 and G_2 of Figure 3.1 have an Euler trail and an Euler tour respectively.

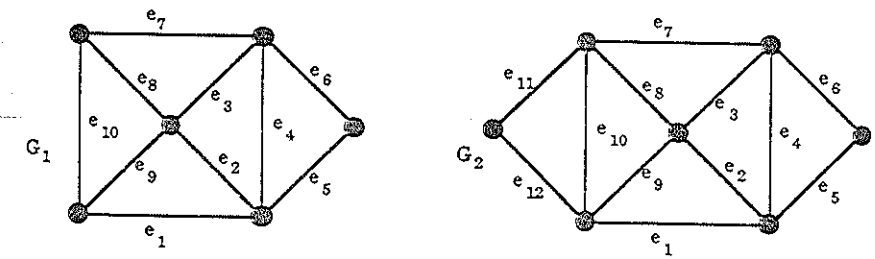


Figure 3.1: G_1 has an Euler trail. G_2 is an Euler graph.

In G_1 an Euler trail, from u to v is given by the sequence of edges $e_1e_2e_3\dots e_9e_{10}$, while in G_2 an Euler tour from u to u is given by $e_1e_2e_3\dots e_{11}e_{12}$. We will see below that G_1 has no Euler tour. Euler trails and tours are, historically, the most famous walks in graph theory. Graphs and graph theory probably began in the early 18th century when the Swiss mathematician, Leonhard Euler, considered the problem of the seven Königsberg bridges. To describe this problem we use Figure 3.2 which gives a simplified map of the Prussian city of Königsberg, as it appeared in the 18th century, showing its site on the banks of the river Pregel:

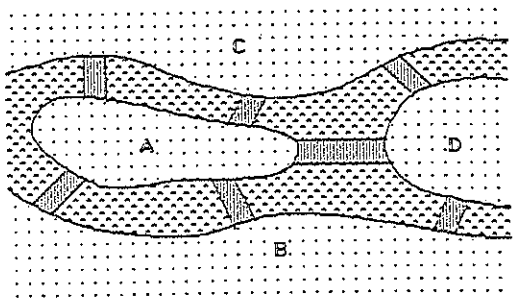


Figure 3.2: A map of Königsberg.

The river was crossed by seven bridges, which connected two islands in the river, shown in Figure 3.2 by A and D, with each other and with the opposite banks B and C. It is said that the townsfolk of Königsberg amused themselves by trying to find a route that crossed each bridge just once. Euler considered this problem by using the graph of Figure 3.3, where each edge represents one of the seven bridges. He then showed [21] the impossibility of such a route by in effect showing that the graph has no Euler trail.

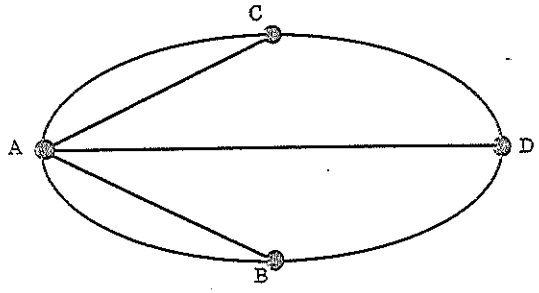


Figure 3.3: A graph representing the bridges of Königsberg.

We will give simple but useful characterizations of Euler graphs and graphs with Euler trails. But first we have a preliminary result.

Theorem 3.1 *Let G be a graph in which the degree of every vertex is at least two. Then G contains a cycle.*

Proof If G is not simple then it contains a cycle since any loop is a cycle of length 1 while a pair of parallel edges gives a cycle of length 2. We now suppose that G is simple. Let v_0 be any vertex of G. Since $d(v_0) \geq 2$, we can choose an edge e_1 with one end v_0 and the other v_1, say. Since $d(v_1) \geq 2$ we can choose an edge e_2 with one end v_1 and the other v_2, say, different from v_0. We repeat this process so that (see Figure 3.4) at the $(i+1)$th stage we have an edge e_i incident with v_i and v_{i+1} and $v_{1+1} \neq v_{i-1}$.

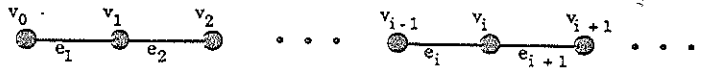


Figure 3.4

Since G has only finitely many vertices, we must eventually choose a vertex which has been chosen before. If v_k is the first such vertex then the walk between the first two occurrences of v_k is a cycle (since the internal vertices of this walk are distinct and also different from v_k as v_k is the first vertex to be repeated).(See Figure 3.5.) □

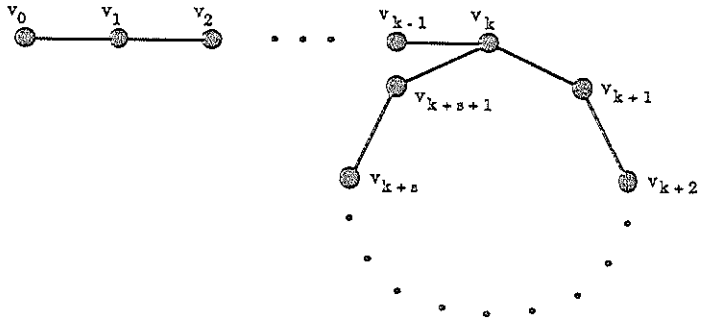


Figure 3.5

Now for our characterization of Euler graphs:

Theorem 3.2 *A connected graph G is Euler if and only if the degree of every vertex is even.*

Proof Suppose that G is Euler. Let C be an Euler tour in G, starting and ending at the vertex u. Then if v is a vertex of G different from u, v must be a vertex on the tour C since u is connected to v (G being connected) and C involves every edge of G. Moreover each time v is met on the tour C, it is entered and left by different edges (because each edge of G only occurs once in C). Thus each occurrence of v in C represents a contribution of 2 to its degree. Thus $d(v)$ is even. Finally, since C begins

and ends with u, the first and last edges of C contribute 2 to the degree of u and any internal occurrence of u on C will (as above) also contribute 2 to $d(u)$. Hence $d(u)$ is also even. Thus the degree of every vertex of G is even, as required.

Conversely, suppose that G is connected and that every vertex is even. We use induction on the number of edges of G to show that G is Euler. Firstly, if there are no edges then, since G is connected, G must consist of a single vertex u, with degree 0. Then the trivial trail $C = u$ involves all the edges of G — since there are none! Thus G, in this case, is Euler.

Now suppose that G does have edges. Then, since G is connected, no vertex of G can have degree 0. Thus, since each vertex is even, each vertex must have degree at least two. Then, by Theorem 3.1, G contains a cycle, C say. If C contains every edge of G then we are finished since then C is an Euler tour. If C does not contain every edge of G then we delete from G every edge in C to form a new (possibly disconnected) graph H which has fewer edges than G, but the same vertex set. Then the vertices of H are still even since any vertex which has had a change in degree has had two distinct incident edges removed from it (because these edges come from the cycle C). (See Figure 3.6.)

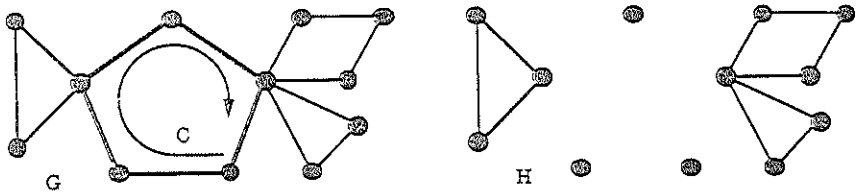


Figure 3.6

Now, assuming the induction hypothesis that the result holds for all graphs with less edges than G, each component of H must be Euler (because the degree of each vertex of H is even). Moreover, since the components were formed by the deletion of C's edges, each component has at least one vertex in common with C. We now obtain an Euler tour for G as follows: start at any vertex of C and go round the edges of C until a vertex belonging to a nonempty component of H is met. Say this vertex is v_1 and then go on an Euler tour in this component starting and ending at v_1. Once back at v_1, resume going round C until the next nonempty component of H is met and repeat the process of going on an Euler tour in this new component. We repeat this procedure eventually going all the way round C back to our starting point, making an Euler tour in each nonempty component of H on the way. We illustrate this in Figure 3.7. Then, since the edges of G are just those of H together with those of C, we have completed an Euler tour of G. By induction the proof is complete. □

Although Euler proved in 1736 the necessity part of Theorem 3.2, i.e., that a connected graph is Euler only if all its vertices are even, surprisingly it was not until 1873 that the sufficiency part was established, by Hierholzer [34]. We refer the reader to Biggs, Lloyd and Wilson [6] for an interesting account of the history of the Königsberg bridges problem (and for a general history of graph theory).

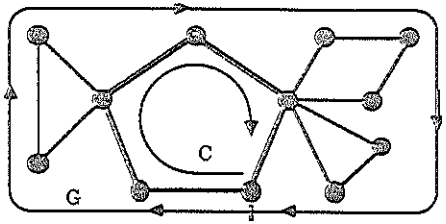


Figure 3.7

We now outline an alternative argument for the second part of the proof of Theorem 3.2, that each vertex of an Euler graph is even. This is a recent simple proof due to Fowler [25].

Thus let G be a connected graph in which every vertex is even. As in the proof of Theorem 3.2, we use induction on the number of edges of G to show that G has an Euler tour. If G has less than three *vertices* then it is an easy matter to see that G has an Euler tour. Hence we may assume that G has at least three vertices. Suppose that G has q edges and, so that we may use induction, that any connected graph H with less than q edges in which every vertex is even is Euler. Since G is connected and has at least three vertices, there exists a vertex v in G joined by two edges to vertices x and y (so that v is distinct from both x and y but x and y are not necessarily distinct). Delete both edges vx and vy from G. Now form a new graph H from this edge deleted subgraph by (i) inserting a new edge xy if $x \neq y$ or (ii) inserting a loop at x if $x = y$. Then it is easy to see that every vertex of H is even and that H has $q - 1$ edges. Thus, by our assumption, if H is connected it must be an Euler graph, say with Euler tour T'. In this case, an Euler tour T for G is then obtained from T' by replacing x, xy and y by x, xv, vy and y.

If H is not connected then it has exactly two connected components, say H_1 and H_2. Moreover, x and y must belong to one component, say H_1, and v to the other, H_2. Our induction assumption provides Euler tours T_1 and T_2 for H_1 and H_2 respectively. We can now construct an Euler tour T for G from T_1 and T_2 by replacing x, xy and y in T_1 by x, xv, T_2, vy and y.

Using the first proof of Theorem 3.2 we can prove another characterisation of Euler graphs, as follows.

Theorem 3.3 *A connected graph G is Euler if and only if G has cycles $C_{(1)}, \ldots, C_{(n)}$ such that every edge of G belongs to exactly one cycle $C_{(i)}$, i.e., G is the union of edge disjoint cycles.*

Proof We leave the proof of this as Exercise 3.1.8. □

Using Theorem 3.2 it is a simple matter to give a characterisation of graphs which have Euler trails:

Theorem 3.4 *A connected graph G has an Euler trail if and only if it has at most two odd vertices, i.e., it has either no vertices of odd degree or exactly two vertices of odd degree.*

Proof Suppose G has an Euler trail. Then, as in the first part of the proof of Theorem 3.2, if v is a vertex different from the origin and terminus of the trail, the degree of v is even. Thus the only possible odd vertices are the origin and terminus of the trail (and if these coincide then in fact we have an Euler tour and every vertex is even). Conversely, suppose that G is connected with at most two odd vertices. If G has no odd vertices then, by Theorem 3.2, G is Euler and so has an Euler trail. This leaves us to treat the case where G has two odd vertices, u and v, say. (G can not have just one odd vertex by Corollary 1.2.)

Now let $G+e$ denote the graph obtained from G by adding in a new edge, joining u to v. Then e increases the degrees of u and v by 1 and so in $G+e$ every vertex is even. Hence, by Theorem 3.2, $G+e$ has an Euler tour say $C = v_0e_1v_1e_2\dots e_nv_n$, involving each edge of $G+e$ exactly once. We may suppose that $e_1 = e, v_0 = u, v_1 = v$ and so $v_n = u$. Then, deleting e from this tour, gives the Euler trail $v_1e_2\dots e_nv_n$, from v to u, in G since it involves each edge of G exactly once. □

Theorems 3.2 and 3.4 can now be applied to the graphs G_2 and G_1 of Figure 3.1 to show that they have an Euler trail, tour respectively. Also the Theorems show that the graph of Figure 3.3, of the Königsberg bridge problem, has no Euler trail since every vertex has odd degree. Euler graphs and, more generally, graphs with Euler trails often appear in books on recreational mathematics or children's puzzle books. For example, a puzzle might ask whether a given diagram can be drawn without lifting one's pencil from the paper and without repeating any lines. In effect this is asking if the corresponding graph has an Euler trail. For example, the first and second diagrams of Figure 3.8 can be drawn in this way, but the third can not.

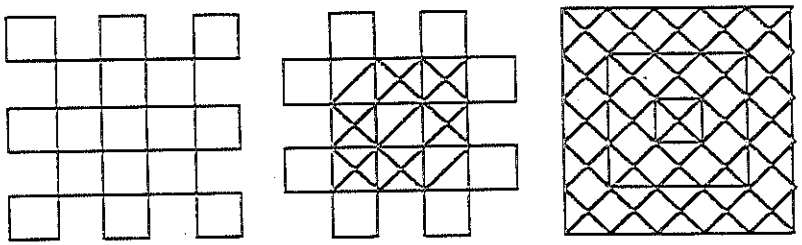


Figure 3.8: Puzzles

The first part of the proof of Theorem 3.4 shows that if a graph G has an Euler trail which is not a tour then the trail must start at one of the odd vertices and end at the other. (This is useful for example when one wants to draw the second diagram above without lifting one's pencil.) We now give an algorithm which constructs an Euler tour in an Euler graph.

Fleury's algorithm

Step 1. Choose any vertex v_0 in the Euler graph G and set $W_0 = v_0$.

Step 2. If the trail $W_i = v_0e_1v_1\dots e_iv_i$ has been chosen (so that $e_1, \dots, e_i$ are all different), choose an edge e_{i+1} different from $e_1, \dots, e_i$ such that

(i) e_{i+1} is incident with v_i and

(ii) unless there is no alternative, e_{i+1} is not a bridge of the edge-deleted subgraph $G - \{e_1, \dots, e_i\}$.

Step 3. Stop if W_i contains every edge of G. Otherwise repeat Step 2.

We illustrate Fleury's algorithm with the Euler graph G of Figure 3.9.

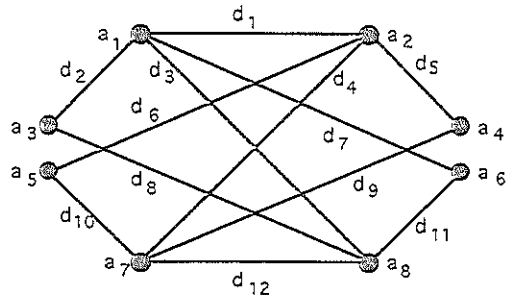
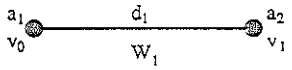


Figure 3.9

Step 1. Choose $v_0 = a_1$. $W_0 = a_1$.

Step 2. Choose edge d_1 for e_1. Figure 3.10 shows W_1.



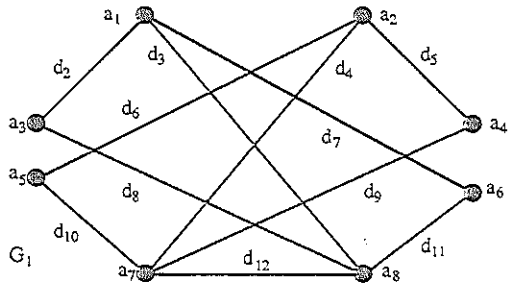


Figure 3.10: The first stage of the Euler tour construction.

Step 2. Choose edge d_5 for e_2. Figure 3.11 shows W_2 and G_2.

Step 2. Choose $e_3 = d_9$. (Although d_9 is a bridge there is no alternative.) Figure 3.11 shows W_3 and G_3.

Step 2. Choose $e_4 = d_4$. (We do not choose d_{12} since it is a bridge in G_3, whereas d_4 is not a bridge.) Figure 3.11 shows W_4 and G_4.

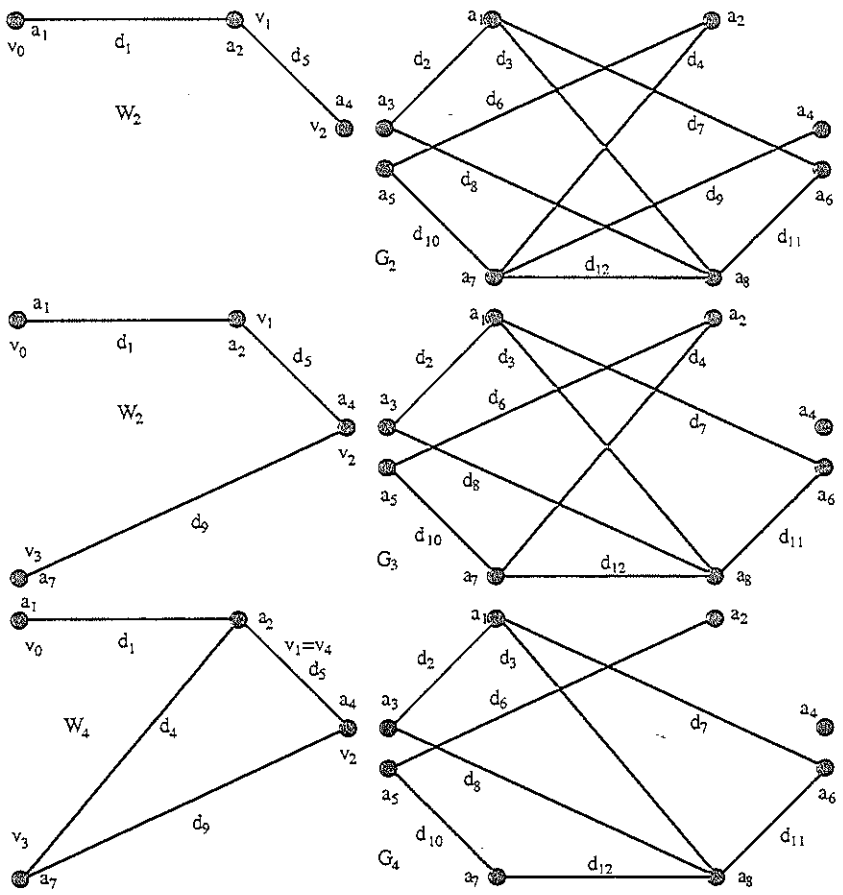


Figure 3.11: The second, third and fourth stages of the Euler tour construction.

Step 2. Choose $e_5 = d_6$ (no choice). (We leave the reader to draw the remaining walks W_i and subgraphs G_i.)

Step 2. Choose $e_6 = d_{10}$ (no choice).

Step 2. Choose $e_7 = d_{12}$ (no choice).

Step 2. Choose $e_8 = d_8$.

Step 2. Choose $e_9 = d_2$ (no choice).

Step 2. Choose $e_{10} = d_3$.

Step 2. Choose $e_{11} = d_{11}$ (although a bridge in G_{10}, no choice).

Step 2. Choose $e_{12} = d_4$ (although a bridge in G_{11}, no choice).

Step 3. Stop.

We have produced the Euler tour:

$$a_1\, d_1\, a_2\, d_5\, a_4\, d_9\, a_7\, d_4\, a_2\, d_6\, a_5\, d_{10}\, a_7\, d_{12}\, a_8\, d_8\, a_3\, d_2\, a_1\, d_3\, a_8\, d_{11}\, a_6\, d_7\, a_1.$$

We now prove that Fleury's algorithm actually does produce an Euler tour.

Theorem 3.5 *Fleury's algorithm produces an Euler tour in an Euler graph G.*

Proof Suppose that the trail $W_i = v_0 e_1 v_1 \dots e_i v_i$ has been chosen by the algorithm and denote the edge-deleted subgraph $G - \{e_1, \dots, e_i\}$ by G_i. Each time a vertex v occurs on the trail W_i which is different from both v_0 and v_i, v has one trail edge going in and one going out, i.e., W_i supplies two to the degree of v each time v occurs. Thus the degree of v in G_i is still even (although possibly 0). Similarly if $v_0 = v_i$ then there is an even number of edges in W_i incident with v_i, namely e_1, e_i and 2 more for each of the other occurrences of v_i. Thus if $v_0 = v_i$ then $d(v_i)$ in G_i is still even. However if $v_0 \neq v_i$ then there is an odd number of edges in W_i incident with v_i, namely e_i and two more for every other occurrence, and so $d(v_i)$ is odd in G_i.

In this latter case, when $v_0 \neq v_i$, if $d(v_i) = 1$ in G_i, i.e., if there is only one edge, e say, in G_i, incident with v_i, then, by Step 2 of the algorithm, we have no option but to choose this edge e to extend W_i to W_{i+1}. In doing so we remove the only edge still incident with v_i to leave v_i isolated. In particular, e is a bridge. (See Figure 3.12.)

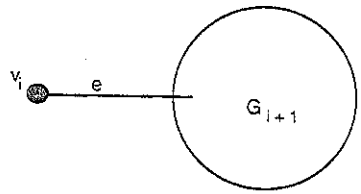


Figure 3.12: $d(v_i) = 1$ in G_i.

If, on the other hand, $d(v_i) \neq 1$ in G_i, then, as we now show, we may choose an edge incident with v_i in G_i which is not a bridge. To prove this it suffices to show that if there are at least two edges in G_i incident with v_i then at most one of these edges is a bridge. Suppose, to the contrary, that e' and f' are two bridges in G_i joining v_i to vertices u, v respectively. Then deleting both e' and f' from G_i drops the degree

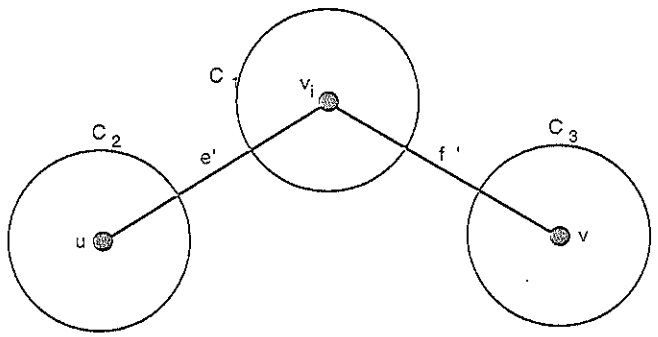


Figure 3.13: e' and f' are bridges in G_i.

of v_i down by 2 so that it is still odd, drops the degree of both u and v by 1, and creates a deleted subgraph $G_i - \{e', f'\}$ with at least three components, v_i being in one component C_1, say, u and v being in two others, say C_2 and C_3. (See Figure 3.13.)

Now the first paragraph of the proof says that the only odd vertices in G_i are v_0 and v_i. Thus if v_0 is different from both u and v, $G_i - \{e', f'\}$ has exactly four odd vertices and so at least one of the above components has exactly one odd vertex — impossible by Corollary 1.2. If on the other hand $v_0 = u$ (or similarly v) then $G_i - \{e', f'\}$ has two odd vertices, namely v_i and v (respectively u). In particular the component C_1 has only one odd vertex, namely v_i — again impossible. This completes the proof that if there is more than one edge incident with v_i then one of these is not a bridge in G_i. Moreover we have shown that in Step 2 of the algorithm, when $v_0 \neq v_i$, if we are forced to choose a bridge e_{i+1} to form the extended trail $W_{i+1} = v_0 e_1 v_1 \dots v_i e_{i+1} v_{i+1}$ then v_i is an isolated vertex in G_{i+1}. In fact, in the process of extending W_i to W_{i+1}, when we disconnect G_i (i.e., choose a bridge) then one of the new components formed is always an isolated vertex.

In the case where $v_0 = v_i$ in W_i, as the first paragraph indicates, each vertex of G_i is of even degree. Thus, if there are still edges in G_i incident with $v_0 = v_i$, then there must be at least two such edges. An argument similar to that above, illustrated by Figure 3.13, then shows that none of these edges are bridges. This shows that throughout the entire implementation of the algorithm if a bridge has to be chosen then doing so leaves behind an isolated vertex and Step 2 can no longer be implemented only if the trail has used all the edges of the graph and has returned to its initial vertex. In other words, the algorithm finishes with an Euler tour of G. □

Exercises for Section 3.1

3.1.1 Determine which of the graphs in Figure 3.14 have (a) Euler trails and (b) Euler tours. For those that have, use Fleury's algorithm to produce such a trail or tour.

3.1.2 (a) Which of the following graphs are Euler?

i. the complete graph K_n, for $n \geq 3$

ii. the n-cube Q_n, for $n \geq 3$ (see Exercise 1.4.7),

iii. the wheel W_n, for $n \geq 4$ (see Exercise 1.6.4).

(b) Which graphs of (a) have Euler trails?

(c) For which $m, n \geq 1$ does the complete bipartite graph $K_{m,n}$ have an Euler tour (or Euler trail)?

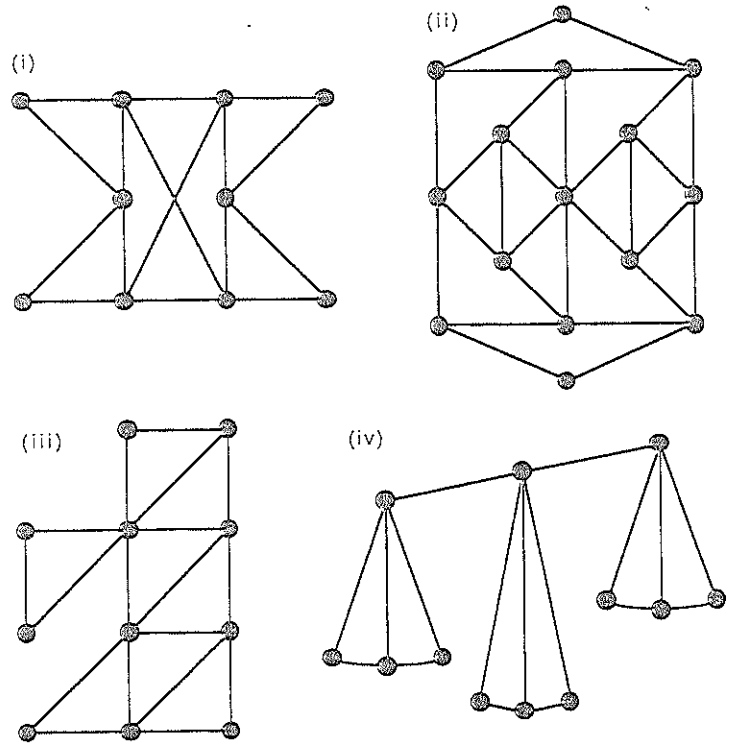


Figure 3.14: Which of these graphs have (a) an Euler tour, (b) an Euler trail?

3.1.3 Let G be a connected graph in which there are exactly four odd vertices. Prove that there are two trails in G such that each edge of G belongs to exactly one of these trails. (Hint: consider the proof of Theorem 3.4.)

3.1.4 Let G be a connected graph in which there are exactly $2k$ odd vertices, where $k \geq 1$. Prove that there are k open trails in G such that each edge of G belongs to exactly one of these trails. Prove also that it is not possible to find $k-1$ open trails with this property.

3.1.5 Using Exercise 3.1.4 determine the minimum number of times you must lift your pencil in order to draw each diagram of Figure 3.15 without repeating a line.

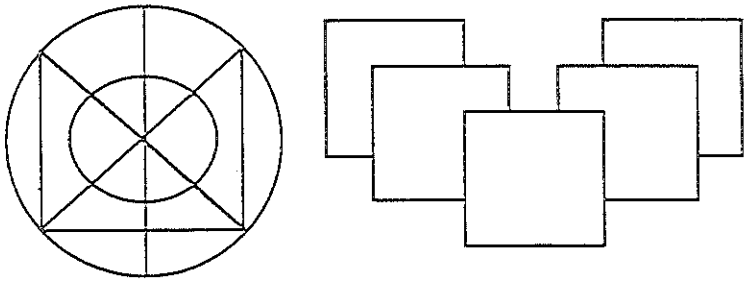


Figure 3.15

3.1.6 Construct a tour for the Königsberg bridges problem so that the number of bridges crossed more than once is as small as possible.

3.1.7 Let G be a nonempty simple graph with edges listed as $e_1, \ldots, e_n$. We define the line graph of G, denoted by $L(G)$, to be the graph with vertices $x_1, \ldots, x_n$ (in one-to-one correspondence with the edges of G) such that x_i and x_j are adjacent if and only if the corresponding edges e_i and e_j are adjacent in G. Figure 3.16 shows a graph G and its line graph $L(G)$.

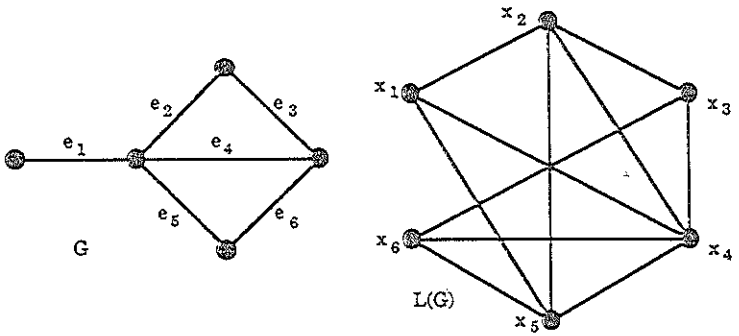


Figure 3.16: A graph G and its line graph $L(G)$.

(a) Prove that if G is Euler then so is $L(G)$.

(b) For each $n \geq 3$ find the line graph of the complete bipartite graph $K_{1,n}$.

(c) Give an example of a simple graph G such that $L(G)$ is Euler but G is not.

3.1.8 Prove Theorem 3.3, i.e., that a connected graph G is Euler if and only if G has cycles $C_{(1)}, \ldots, C_{(n)}$ such that every edge of G belongs to exactly one cycle $C_{(i)}$, i.e., G is the union of edge disjoint cycles. (Hint: try to modify the proof of Theorem 3.2.)

3.1.9 An Euler graph G is called **randomly traceable** from a vertex v if every trail in G starting at v can be extended to an Euler tour, starting and ending at v.

(a) Show that the Euler graph of Figure 3.17 is randomly traceable from the vertex v indicated, but not from any other vertex.

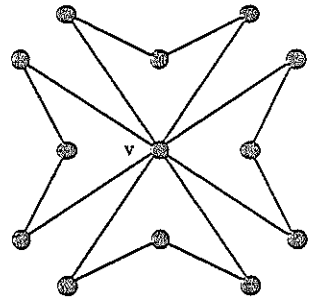


Figure 3.17: Show that this graph is randomly traceable from v but not from any other vertex.

(b) Give an example of an Euler graph which is not randomly traceable from any of its vertices.

(c) Give an example of an Euler graph which is randomly traceable from all of its vertices.

(d) Prove that the Euler graph G is randomly traceable from its vertex v if and only if v lies on every cycle of G.

3.1.10 Here is another algorithm, due to Hierholzer, which also produces an Euler tour in an Euler graph G. Its method is to start with any closed trail in G and "attach" to it, systematically, "detour" trails until all edges of G are used up. (In what follows, $E(H)$ denotes the set of edges of a subgraph H.)

Hierholzer's algorithm

Step 1. Choose any vertex v in G and choose any closed trail W_0 in G starting and ending at v. Set $i = 0$. (i is a counter.)

Step 2. If $E(W_i) = E(G)$ stop, since then W_i is an Euler tour of G.
Otherwise choose a vertex v_i on W_i which is incident with an edge in G which is not in W_i.
Now choose a closed trial W_i^* in the subgraph $G - E(W_i)$, starting at the vertex v_i. (W_i^* is a "detour" trail.)

Step 3. Let W_{i+1} be the closed trail consisting of the edges of both W_i and W_i^* obtained by starting at the vertex v, traversing the trail W_i until v_i is reached, then traversing the closed trail W_i^* and, on returning to v_i, completing the rest of the trail W_i.
Now set $i = i + 1$ and return to Step 2.

Use Hierholzer's algorithm to produce an Euler tour for the graph of Figure 3.9 starting with $W_0 = a_1a_2a_6a_1$.

3.2 The Chinese Postman Problem

Before starting on his delivery route, a postman must pick up his letters at the post office, then he must deliver letters along each street on his route, and finally he must return to the post office to return all undelivered letters. Wishing to conserve energy, every postman would like to cover his route with as little walking as possible. In nongraph terms, the postman problem is how to cover all the streets in the route and return back to the starting point with as little travelling as possible. It is usually referred to as the Chinese Postman Problem (CPP for short) because the first work published on the problem was by a Chinese mathematician, Kuan, in 1962 [39].

In graph-theoretical terms the problem is dealt with as follows. We construct a weighted graph G where each edge represents a street in the postman's route, each vertex represents a junction of streets and the weight assigned to each edge represents the length of the street between junctions. If we define the weight of a tour $v_0e_1v_1e_2 \dots e_nv_0$ in G to be the sum of the weights of its edges, i.e., $w(e_1) + w(e_2) + \cdots w(e_n)$, the Chinese postman problem is just that of finding a tour of minimum weight in a weighted connected graph with non-negative weights. (Recall that a tour involves each edge of G at least once.)

If the weighted graph G so constructed is Euler then any Euler tour of G is a tour of minimum weight since it involves each edge of G once and only once. Thus in practice we may use Fleury's algorithm to produce such a tour, i.e., a solution to the CPP.

If G is not Euler then any tour in G has to involve some edges more than once. For example, in the weighted graph of Figure 3.18, an optimal tour, i.e., a minimal weighted tour is given by

$$v_1\ v_2\ v_3\ v_6\ v_2\ v_3\ v_6\ v_7\ v_5\ v_4\ v_7\ v_5\ v_1\ v_4\ v_3\ v_1$$

where the edges v_2v_3, v_3v_6 and v_5v_7 are each used twice. Of course, at this stage we have no easy way of telling that this tour is optimal.

In order to look at a particular case of the non-Eulerian situation we introduce the process of duplication of an edge.

> An edge e is said to be duplicated when its ends are joined by a new edge with the same weight $w(e)$ as e.

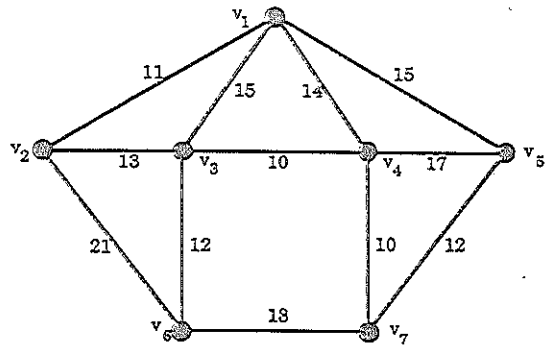


Figure 3.18: A postman's delivery area.

For example, by duplicating the edges v_4v_6, v_6v_8,v_8v_7 and v_7v_5 in the graph G of Figure 3.19, we produce the supergraph G^* of G:

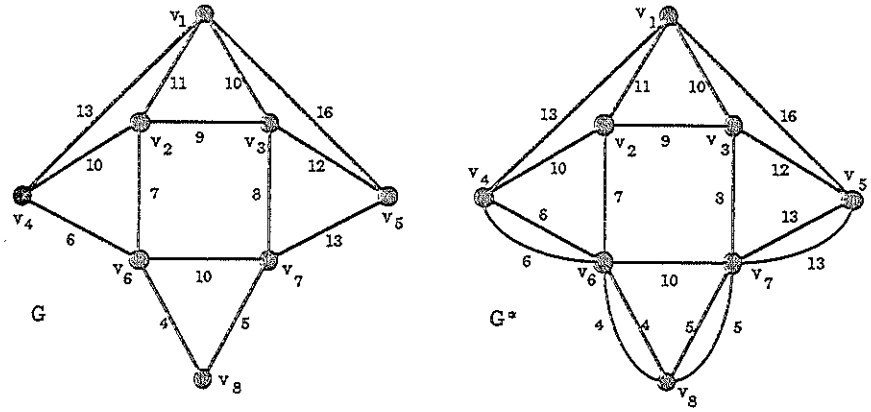


Figure 3.19: A graph G with an Euler supergraph G^* obtained by duplicating edges.

The Chinese Postman Problem may now be rephrased as follows.
Given a connected weighted graph G with non-negative weights,
(i) find, by duplicating edges if necessary, an Euler weighted supergraph G^* of G such that the sum of the weights of the duplicated edges is as small as possible, i.e.,

$$\sum_{e \in E(G^*) - E(G)} w(e)$$

is at a minimum where $E(G^*) - E(G)$ is the set of edges in G^* but not in G, and
(ii) find an Euler tour in G^*.

To see that this is just the same as the Chinese Postman Problem just note that an optimal tour of G in which an edge e is repeated n times corresponds to an (optimal) tour of G^* where the edge e is duplicated $n-1$ times.

At this stage we will only consider a special case of the problem, namely when G *has exactly two vertices of odd degree.*

Suppose that G has exactly two vertices, u and v, of odd degree. Let G^* be an Euler supergraph of G obtained by duplicating edges. Then, since u and v now have even degree in G^*, the duplicated edges will form a walk from u to v. (There must be a duplicated edge out of u to give u its even degree. If this edge, e_1 say, goes to vertex v_1 then the degree of v_1 increases by 1. If $v_1 = v$ then v now has even degree and the edge e_1 gives a walk (in fact a path) from u to v. If $v_1 \neq v$ then the edge e_1 has changed v_1's even degree in G to odd so there must be a second edge, e_2 say, going out of v_1 to give v_1 even degree in G^*. Continuing in this way we eventually reach v by a sequence of duplicated edges since, in order to make v have even degree in G^*, v must have a duplicated edge incident.)

Clearly to satisfy criterion (i), i.e., to optimise G^*, the length of such a walk of duplicated edges from u to v in G^* should be as small as possible. Thus in the case where G has exactly two vertices, u and v, of odd degree, to solve (i) we just have to find a shortest path in G from u to v and then duplicate the edges which form this path to get the Euler supergraph G^*.

To illustrate the complete procedure we use the graph G of Figure 3.19, which has two odd vertices v_4 and v_5. First we apply Dijkstra's algorithm to find a shortest path from v_4 to v_5. Following the presentation of the algorithm given in Section 2.5, we get the table:

	vertices v							T
v_4,	v_1,	v_2,	v_3,	v_6,	v_7,	v_8,	v_5	
[0]	∞	∞	∞	∞	∞	∞	∞	$\{v_4, v_1, v_2, v_3, v_6, v_7, v_8, v_5\}$
	13	10	∞	[6]	∞	∞	∞	$\{\ \ v_1, v_2, v_3, v_6, v_7, v_8, v_5\}$
	13	10	∞		16	10	∞	$\{\ \ v_1, v_2, v_3, \ \ v_7, v_8, v_5\}$
	13		19		16	[10]	∞	$\{\ \ v_1, \ \ v_3, \ \ v_7, v_8, v_5\}$
	13		19		15		∞	$\{\ \ v_1, \ \ v_3, \ \ v_7, \ \ v_5\}$
			19		[15]		29	$\{\ \ \ \ v_3, \ \ v_7, \ \ v_5\}$
			19				29	$\{\ \ \ \ v_3, \ \ \ \ \ \ v_5\}$
							[29]	$\{\ \ \ \ \ \ \ \ \ \ \ \ v_5\}$

Since $v_4\ v_6\ v_8\ v_7\ v_5$ is a shortest path from v_4 to v_5, (as indicated in the table), duplicating each of the edges in this path gives the Euler weighted supergraph G^* shown in Figure 3.19.

It is now left to carry out (ii), i.e., find an Euler tour in G^*. We use Fleury's algorithm starting at vertex v_1 (the Post Office). Then an Euler tour is given by

$$v_1\ v_4\ v_6\ v_8\ v_7\ v_5\ v_1\ v_2\ v_4\ v_6\ v_8\ v_7\ v_5\ v_3\ v_2\ v_6\ v_7\ v_3\ v_1.$$

It has length 162. Fleury's algorithm can of course provide several different Chinese postman tours apart from this one.

Exercise for Section 3.2

3.2.1 Solve the Chinese Postman Problem for the graphs of Figure 3.20.

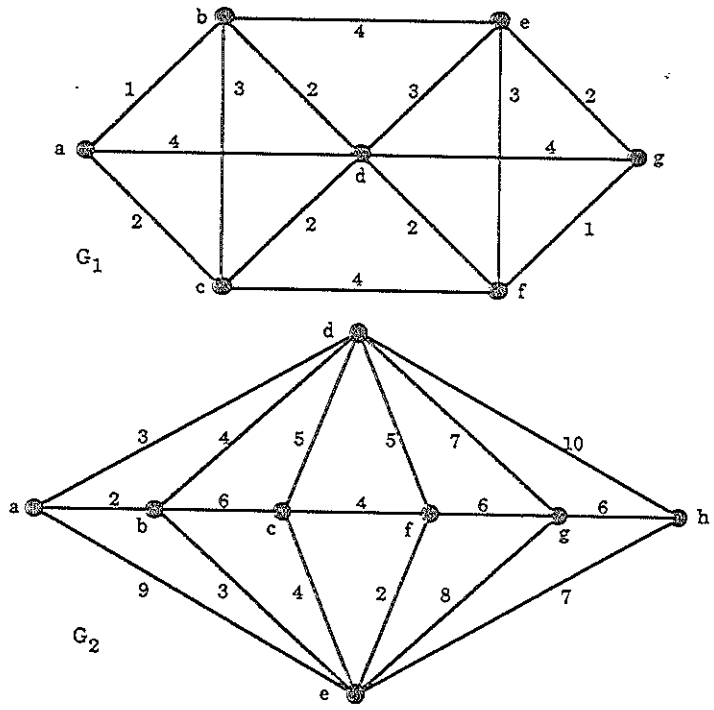


Figure 3.20

3.3 Hamiltonian Graphs

A **Hamiltonian path** in a graph G is a path which contains every vertex of G.

Since, by definition (see Section 1.6), no vertex of a path is repeated, this means that a Hamiltonian path in G contains every vertex of G once and only once.

A Hamiltonian cycle (or Hamiltonian circuit) in a graph G is a cycle which contains every vertex of G.

Since, by definition (again see Section 1.6), no vertex of a cycle is repeated apart from the final vertex being the same as the first vertex, this means that a Hamiltonian cycle in G with initial vertex v contains every other vertex of G precisely once and then ends back at v.

A graph G is called Hamiltonian if it has a Hamiltonian cycle.

By simply deleting the last edge of a Hamiltonian cycle we get a Hamiltonian path. However a non-Hamiltonian graph may possess a Hamiltonian path, i.e., Hamiltonian paths cannot always be used to form Hamiltonian cycles. For example, in Figure 3.21, G_1 has no Hamiltonian path (and so no Hamiltonian cycle), G_2 has the Hamiltonian path $a\ b\ c\ d$ but has no Hamiltonian cycle, while G_3 has the Hamiltonian cycle $a\,b\,d\,c\,a$.

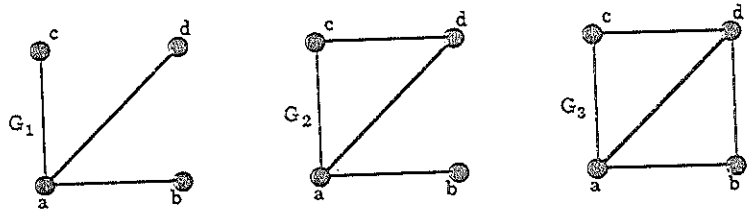


Figure 3.21: G_1 has no Hamiltonian path, G_2 has a Hamiltonian path but no Hamiltonian cycle, while G_3 has a Hamiltonian cycle.

Hamiltonian graphs are named after Sir William Hamilton, an Irish mathematician (1805–1865), who invented a puzzle, called the Icosian game, which he sold for 25 guineas to a games manufacturer in Dublin. The puzzle involved a dodecahedron on which each of the 20 vertices was labelled by the name of some capital city in the world. The object of the game was to construct, using the edges of the dodecahedron, a tour of all the cities which visited each city exactly once, beginning and ending at the same city. In other words, one had essentially to form a Hamiltonian cycle in the graph corresponding to the dodecahedron. We show such a cycle, using bolder lines, in Figure 3.22.

Clearly the n-cycle C_n with n distinct vertices (and n edges) is Hamiltonian. Moreover, given any Hamiltonian graph G, then, if G' is a supergraph of G obtained by adding in new edges between vertices of G, G' will also be Hamiltonian, since any Hamiltonian cycle in G will also be a Hamiltonian cycle in G'. In particular since K_n, the complete graph on n vertices, is such a supergraph of an n-cycle, K_n is Hamiltonian.

A graph G will be Hamiltonian if and only if its underlying simple graph is Hamiltonian since if G is Hamiltonian then any Hamiltonian cycle in G will remain a Hamiltonian cycle in the underlying simple graph of G (provided we delete the appropriate parallel edges). Conversely, if the underlying simple graph of a graph G

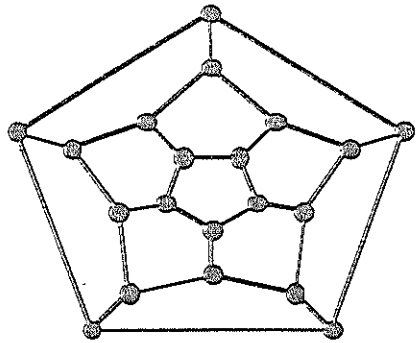


Figure 3.22: A Hamiltonian cycle in the graph of the dodecahedron.

is Hamiltonian then G will also be, because of the remarks of the previous paragraph. For this reason one usually only considers the Hamiltonian property for simple graphs.

Given a simple graph G with n vertices, since G is a subgraph of the complete graph K_n, we can construct step-by-step simple supergraphs of G to eventually get K_n, simply by adding in an extra edge at each step between two vertices that are not already adjacent. We illustrate this in Figure 3.23.

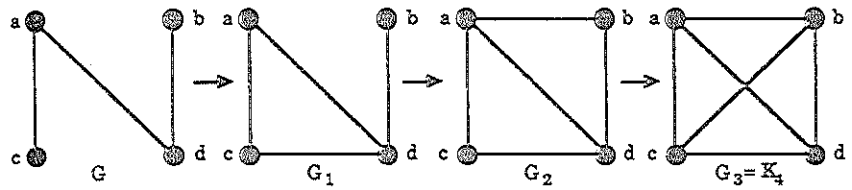


Figure 3.23: A build-up to K_4.

If, moreover, we start with a graph G that is not Hamiltonian, then, since the final outcome of the procedure is the Hamiltonian graph K_n, at some stage during the procedure we change from a non-Hamiltonian graph to a Hamiltonian graph. For example, the non-Hamiltonian graph G_1 above is followed by the Hamiltonian graph G_2. Notice that since supergraphs of Hamiltonian graphs are Hamiltonian, once a Hamiltonian supergraph is reached in the procedure, all the subsequent supergraphs are Hamiltonian. This discussion leads to the following definition.

A simple graph G is called maximal non-Hamiltonian if it is not Hamiltonian but the addition to it of any edge connecting two non-adjacent vertices forms a Hamiltonian graph.

For example, G_1 of Figure 3.21 is maximal non-Hamiltonian since the addition of the edge ab gives the Hamiltonian G_2 as shown, while the other possibility, the addition

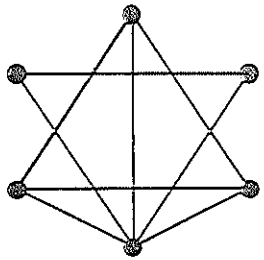


Figure 3.24: A maximal non-Hamiltonian graph.

of the edge bd, also gives a Hamiltonian graph (with Hamiltonian cycle $b\ d\ a\ c\ b$). Similarly the graph of Figure 3.24 is also maximal non-Hamiltonian:

Because of the stepwise procedure described above any non-Hamiltonian graph with n vertices will be a subgraph of a maximal non-Hamiltonian graph with n vertices. We use this to prove the following theorem, due to Dirac [18].

Theorem 3.6 (Dirac, 1952) *If G is a simple graph with n vertices, where $n \geq 3$, and the degree $d(v) \geq n/2$ for every vertex v of G, then G is Hamiltonian.*

Proof We suppose that the result is false. Then, for some value $n \geq 3$, there is a non-Hamiltonian graph in which every vertex has degree at least $n/2$. Any spanning supergraph, i.e., with precisely the same vertex set, also has every vertex with degree at least $n/2$, since any proper supergraph of this form is obtained by introducing more edges. Thus there will be a maximal non-Hamiltonian graph G with n vertices and $d(v) \geq n/2$ for every v in G. Using this G we obtain a contradiction.

G can not be complete, since K_n is Hamiltonian. Thus there are two nonadjacent vertices u and v in G. Let $G+uv$ denote the supergraph of G obtained by introducing an edge from u to v. Then, since G is maximal non-Hamiltonian, $G+uv$ must be Hamiltonian. Also, if C is a Hamiltonian cycle of $G+uv$, then it must contain the edge uv (since otherwise it would be a Hamiltonian cycle in G). Thus, choosing such a C, we may write $C = v_1v_2 \ldots v_nv_1$ where $v_1 = u, v_n = v$ (and the edge v_nv_1 is just vu, i.e., uv). Now let

$$S = \{v_i \in C : \text{ there is an edge from } u \text{ to } v_{i+1} \text{ in } G\} \text{ and}$$

$$T = \{v_j \in C : \text{ there is an edge from } v \text{ to } v_j \text{ in } G\}.$$

Then $v_n \notin T$, since otherwise there would be an edge from v to $v_n = v$, i.e., a loop, impossible because G is simple. Also $v_n \notin S$ (interpreting v_{n+1} as v_1), since otherwise we would again get a loop, this time from u to $v_1 = u$. Thus $v_n \notin S \cup T$. Then, letting $|S|$, $|T|$ and $|S \cup T|$ denote the number of elements in S, T, and $S \cup T$ respectively, we get

$$|S \cup T| < n \tag{3.1}$$

Also, for every edge incident with u there corresponds precisely one vertex v_i in S. Thus

$$|S| = d(u) \tag{3.2}$$

Similarly

$$|T| = d(v) \tag{3.3}$$

Moreover, if v_k is a vertex belonging to both S and T, then there is an edge e joining u to v_{k+1} and an edge f joining v to v_k. This would give

$$C' = v_1\ v_{k+1}\ v_{k+2}\ \cdots\ v_n\ v_k\ v_{k-1}\ \cdots\ v_2\ v_1.$$

as a Hamiltonian cycle in G, (see Figure 3.25), a contradiction since G is non-Hamiltonian.

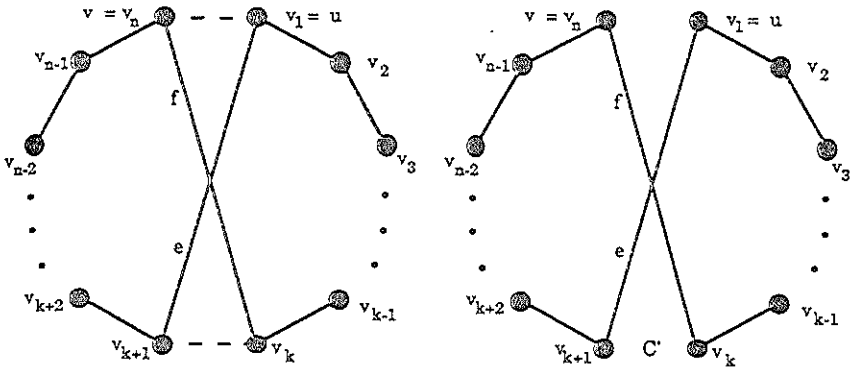


Figure 3.25

This shows that there is no vertex v_k in $S \cap T$, i.e., $S \cap T = \emptyset$. Thus $|S \cup T| = |S| + |T|$. Hence, by (3.1), (3.2) and (3.3) above,

$$d(u) + d(v) = |S| + |T| = |S \cup T| < n. \tag{3.4}$$

This is impossible since in G, $d(u) \geq n/2$ and $d(v) \geq n/2$, and so $d(u) + d(v) \geq n$. This contradiction tells us that we have wrongly assumed the result to be false. □

We now use the ideas of the above proof to present some results on Hamiltonian graphs by Bondy and Chvatal [8].

Theorem 3.7 *Let G be a simple graph with n vertices and let u and v be non-adjacent vertices in G such that*

$$d(u) + d(v) \geq n.$$

Let $G+uv$ denote the supergraph of G obtained by joining u and v by an edge. Then G is Hamiltonian if and only if $G+uv$ is Hamiltonian.

Proof Suppose that G is Hamiltonian. Then, as noted earlier, the supergraph $G+uv$ must also be Hamiltonian.

Conversely, suppose that $G+uv$ is Hamiltonian. Then, if G is not Hamiltonian, just as in the proof of Theorem 3.6 we obtain the inequality $d(u) + d(v) < n$. However, by hypothesis, $d(u) + d(v) \geq n$. Hence G must be Hamiltonian also, as required. □

Motivated by Theorem 3.7 we now define what we mean by the closure $c(G)$ of a simple graph G.

> Let G be a simple graph. If there are two nonadjacent vertices u_1 and v_1 in G such that $d(u_1) + d(v_1) \geq n$ in G, join u_1 and v_1 by an edge to form the supergraph G_1. Then, if there are two nonadjacent vertices u_2 and v_2 such that $d(u_2) + d(v_2) \geq n$ in G_1, join u_2 and v_2 by an edge to form the supergraph G_2. Continue in this way, recursively joining pairs of nonadjacent vertices whose degree sum is at least n until no such pair remains. The final supergraph thus obtained is called the closure of G and is denoted by $c(G)$.

We give an example of the closure operation in Figure 3.26. For this example, $c(G) = K_7$.

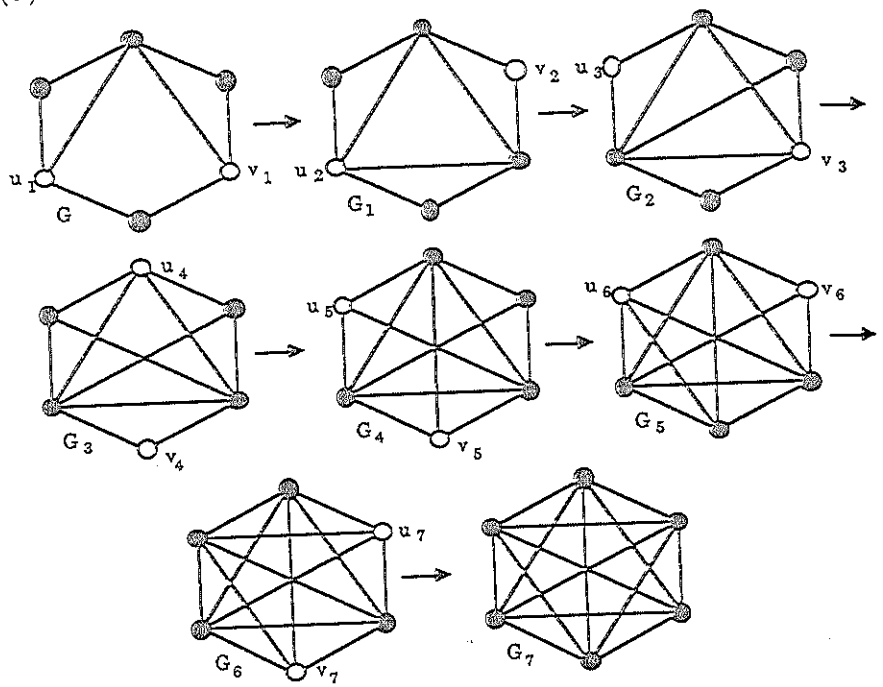


Figure 3.26: Here the closure operation joins the pairs of vertices shown in white and the closure is reached after seven such joins.

On the other hand, for the graph G on 7 vertices of Figure 3.27, $d(u) + d(v) < 7$ for any pair u, v of nonadjacent vertices in G and so the closure operation does not get off the ground, i.e., $c(G) = G$.

Notice that in the example of Figure 3,26 there were often various choices available of pairs of nonadjacent vertices u, v with $d(u) + d(v) \geq n$. Thus the closure procedure could have been carried out in several different ways. The question arises as to whether each different way gives the same result, i.e., do we always end with the same $c(G)$? Yes, we do — we omit the details here. (The interested reader is referred to Lemma 4.4.2 of Bondy and Murty [7].)

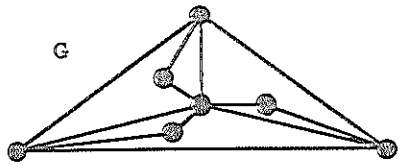


Figure 3.27: $c(G) = G$.

The importance of $c(G)$ is given in the following result:

Theorem 3.8 (Bondy and Chvatal, 1976) *A simple graph G is Hamiltonian if and only if its closure $c(G)$ is Hamiltonian.*

Proof Since $c(G)$ is a supergraph of G, if G is Hamiltonian then $c(G)$ must be Hamiltonian.

Conversely, suppose that $c(G)$ is Hamiltonian. Let $G, G_1, G_2, \ldots, G_{k-1}, G_k = c(G)$ be the sequence of graphs obtained by performing the closure procedure on G. Since $c(G) = G_k$ is obtained from G_{k-1} by setting $G_k = G_{k-1} + uv$, where u, v is a pair of nonadjacent vertices in G_{k-1} with $d(u)+d(v) \geq n$, it follows by Theorem 3.7 that G_{k-1} is Hamiltonian. Similarly G_{k-2}, so $G_{k-3}, \ldots$, so G_1, and so G must be Hamiltonian, as required. □

Corollary 3.9 *Let G be a simple graph on n vertices, with $n \geq 3$. If $c(G)$ is complete, i.e., if $c(G) = K_n$, then G is Hamiltonian.*

Proof This is immediate from the Theorem since any complete graph is Hamiltonian. □

For example, for the graph G of Figure 3.26 we got $c(G)$ as the complete graph K_7 and so, by the Corollary, it follows that G is Hamiltonian.

Unfortunately, the closure operation is not always helpful in determining if a graph is Hamiltonian. For example, $c(G) = G$ for the graph G of Figure 3.27 so here the operation provides no additional information.

Although the closure operation tells us that the graph G of Figure 3.26 is Hamiltonian, this is obvious from the drawing of G. For this reason we look at a less obvious example G in Figure 3.28.

We perform the closure operation on G in Figures 3.28 and 3.29, illustrating the nonadjacent pair of vertices u, v with $d(u) + d(v) \geq n = 9$ we use at each step by white dots. Since the final graph G_{15} in our closure construction is the complete graph K_9, we may conclude from the Corollary that our initial graph G is Hamiltonian.

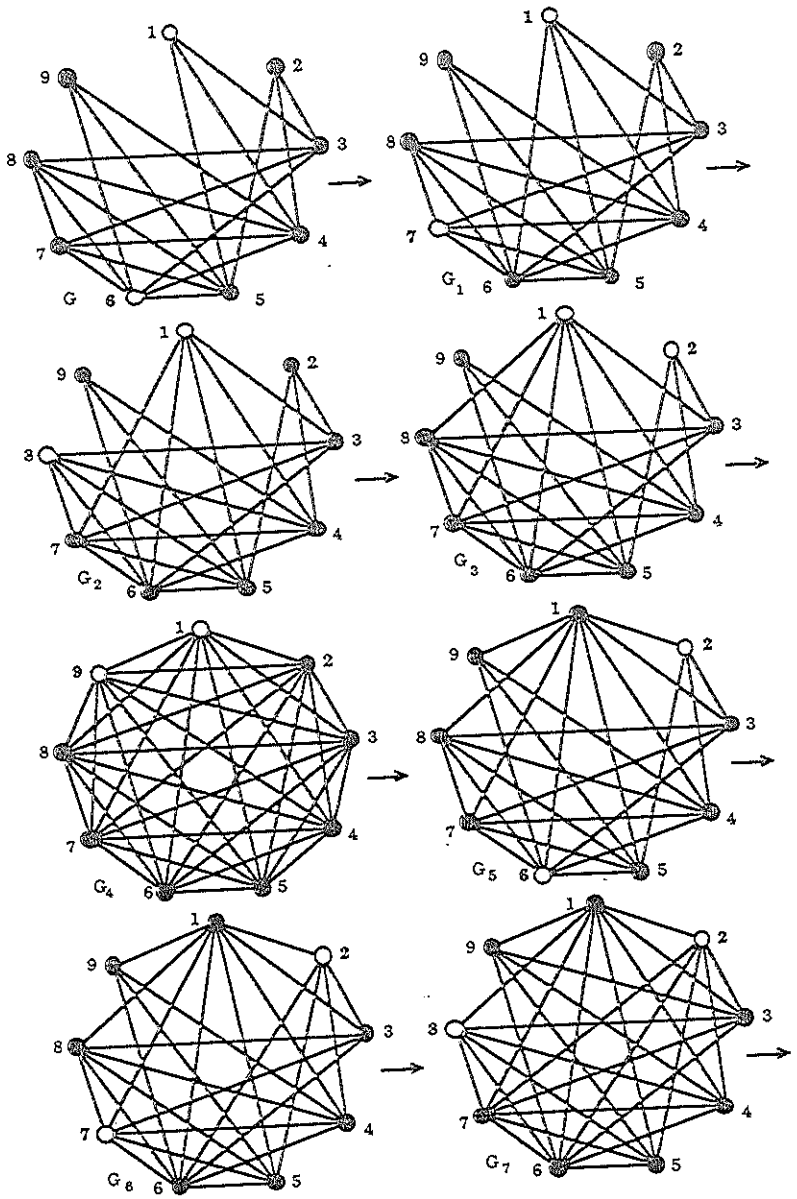


Figure 3.28: The closure operation on G. In G_5 vertex 1 has reached (maximum possible) degree 8, while vertex 6 reaches degree 8 in G_6.

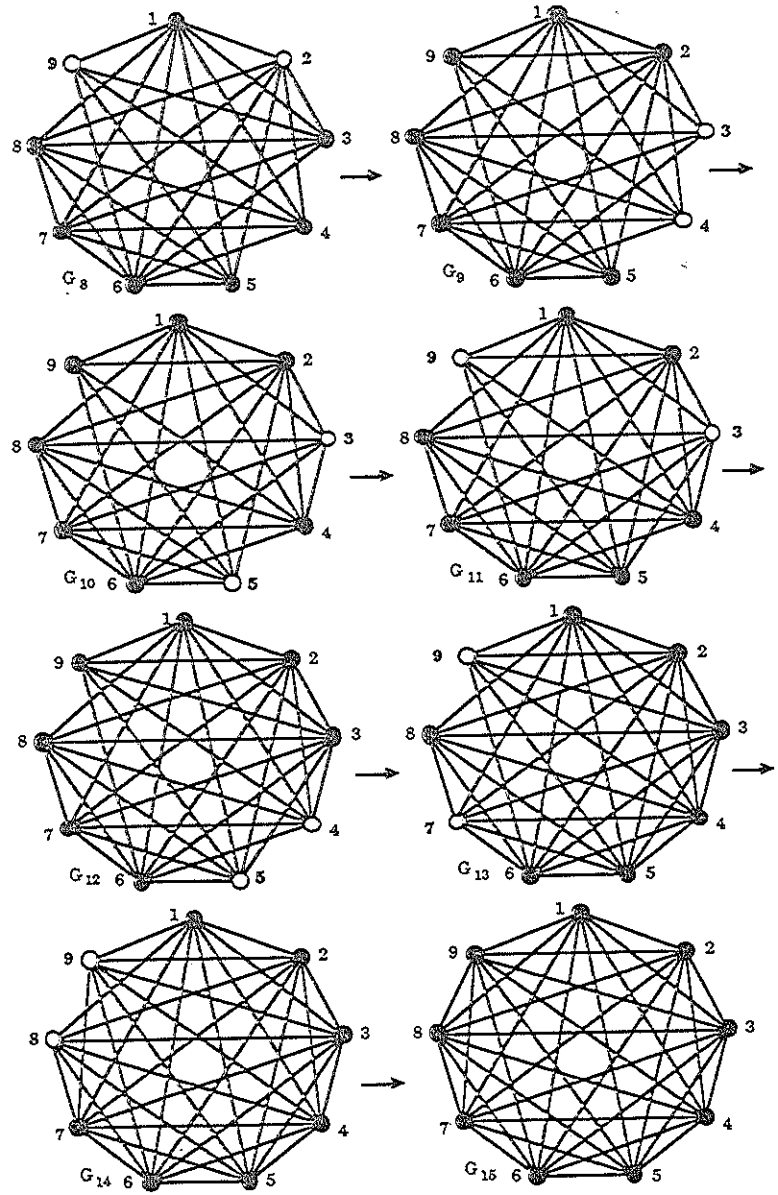


Figure 3.29: The closure operation continued. Vertices 2 and 3 reach degree 8 in G_9 and G_{12} respectively, vertices 4 and 5 reach degree 8 in G_{13}, vertex 7 reaches degree 8 in G_{14} and, finally, vertices 8 and 9 reach degree 8 in G_{15}.

Exercises for Section 3.3

3.3.1 Show that the graph G_1 of Figure 3.30 is Hamiltonian and that the graph G_2 has a Hamiltonian path but has not a Hamiltonian cycle. (Try looking at the vertices of degree two.)

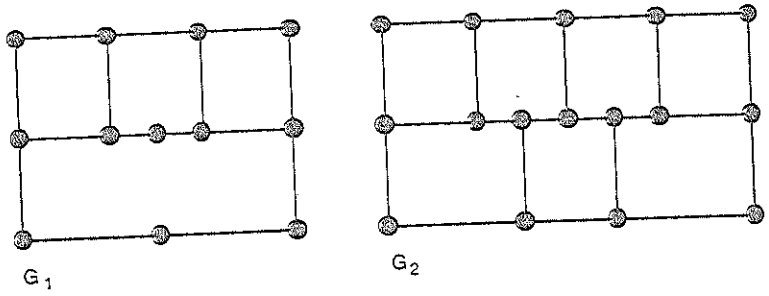


Figure 3.30

3.3.2 Characterise all simple Euler graphs having an Euler tour which is also a Hamiltonian cycle.

3.3.3 Let G be a bipartite graph with bipartition $V = X \cup Y$.

(a) Show that if G is Hamiltonian then $|X| = |Y|$.

(b) Show that if G is not Hamiltonian but has a Hamiltonian path then either $|X| = |Y|$ or $|X| = |Y| \pm 1$.

3.3.4 Prove that the wheel W_n is Hamiltonian for every $n \geq 4$.

3.3.5 Prove that the n-cube Q_n is Hamiltonian for each $n \geq 2$.

3.3.6 Prove that the graphs (i) and (ii) of Figure 3.14 are Hamiltonian but (iii) and (iv) are not. Prove that graph (iii) does have a Hamiltonian path though, but graph (iv) does not.

3.3.7 Let G be a Hamiltonian graph. Show that G does not have a cut vertex.

3.3.8 Let G be a Hamiltonian graph and let S be a proper subset of vertices of G. Prove that $\omega(G - S) \leq |S|$. (This generalises the previous Exercise.)

3.3.9 Show, by giving an example, that the condition "$d(v) \geq n/2$" in Dirac's Theorem (Theorem 3.6) can not be changed to "$d(v) \geq (n-1)/2$".

3.3.10 Let H be an Euler graph and let $G = L(H)$, the line graph of H. (See Exercise 3.1.7.) Prove that G is Hamiltonian.

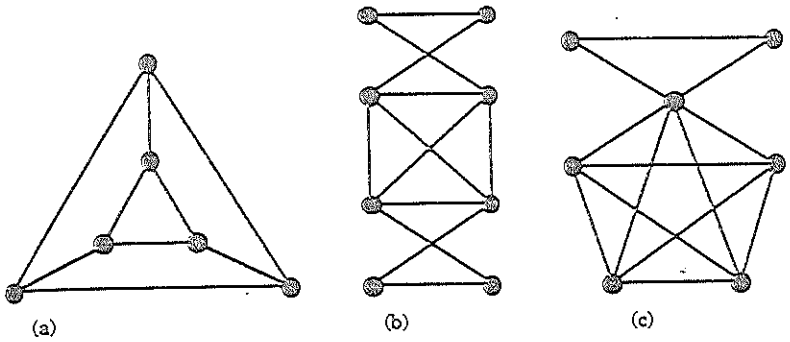


Figure 3.31: Find the closure of each of these graphs.

3.3.11 Find the closure $c(G)$ for each of the graphs of Figure 3.31. Which of these graphs are Hamiltonian?

3.3.12 Prove that, for each $n \geq 1$, the complete tripartite graph $K_{n,2n,3n}$ is Hamiltonian but $K_{n,2n,3n+1}$ is not Hamiltonian. (See Exercise 1.6.13 for the definition of $K_{r,s,t}$.)

3.3.13 In a Hamiltonian graph G two Hamiltonian cycles C and C' are considered to be the same if C is a cyclic rotation of C' or a cyclic rotation of the reverse of C'. Thus, for example, if $C = v_1 v_2 v_3 v_4 v_1$ is a Hamiltonian cycle in G we consider it to be the same as the Hamiltonian cycles $C_{(1)} = v_2 v_3 v_4 v_1 v_2$, $C_{(2)} = v_3 v_4 v_1 v_2 v_3$, $C_{(3)} = v_4 v_1 v_2 v_3 v_4$, $C_{(4)} = v_1 v_4 v_3 v_2 v_1$, $C_{(5)} = v_4 v_3 v_2 v_1 v_4$, $C_{(6)} = v_3 v_2 v_1 v_4 v_3$ and $C_{(7)} = v_2 v_1 v_4 v_3 v_2$.

(a) Prove that the complete graph K_n has $(n-1)!/2$ different Hamiltonian cycles.

(b) How many different Hamiltonian cycles does $K_{n,n}$ have?

3.3.14 There are n guests at a dinner party, where $n \geq 4$. Any two of these guests know, between them, all the other $n-2$. Prove that the guests can be seated round a circular table so that each one is sitting between two people they know.

3.3.15 Let G be a simple k-regular graph with $2k-1$ vertices. Prove that G is Hamiltonian.

3.3.16 Let G_1 and G_2 be two simple graphs and let G denote their join $G_1 + G_2$. (See Exercise 1.5.5.) Prove that if G_1 is Hamiltonian and has at least as many vertices as G_2 has then G is also Hamiltonian.

3.4 The Travelling Salesman Problem

Suppose a travelling salesman's territory includes several towns with roads connecting certain pairs of these towns. His job requires him to visit each town. Is it possible for him to plan a round trip by car enabling him to visit each of the towns exactly once and, if such a trip is possible, can he plan one which minimises the total distance travelled?

We can represent the salesman's territory by a weighted graph G where the vertices correspond to the towns and two vertices are joined by a weighted edge if and only if there is a road connecting the corresponding towns which does not pass through any of the other towns, the edge's weight representing the length of the road between the towns. The question posed above becomes:

Is G a Hamiltonian graph and if so can we construct a Hamiltonian cycle of minimum weight (length)?

This problem is known as the Travelling Salesman Problem and we shall refer to a Hamiltonian cycle of minimum weight as an **optimal circuit**. There are two difficulties that arise with the problem:

(1) It is sometimes difficult to determine if a graph is Hamiltonian (since there is no easy characterization of Hamiltonian graphs such as Theorem 3.2 for Euler graphs).

(2) Given a weighted graph G which is Hamiltonian there is no easy or efficient algorithm for finding an optimal circuit in G, in general.

Here we look at a special case, namely when *the weighted graph is complete*, i.e., it is simple and every pair of distinct vertices is joined by an edge. Even with this special case we will have to be content with looking at two algorithms which produce "reasonably good" solutions, i.e., they produce Hamiltonian circuits which are quite short in length but there is no guarantee that they are the shortest possible.

In the first algorithm, known as the **two-optimal method**, we begin by choosing a Hamiltonian cycle C of the complete weighted graph G. Thereafter we perform a sequence of modifications to C in the hope of finding a cycle of smaller weight.

In more detail, let $C = v_1\ v_2 \dots v_n\ v_1$ be the initial Hamiltonian cycle in our complete graph G. Then, for every pair of numbers i, j such that $1 < i+1 < j \leq n$, we can form a new Hamiltonian cycle C_{ij} from C given by

$$C_{ij} = v_1\ v_2 \dots v_i\ v_j\ v_{j-1} \dots v_{i+1}\ v_{j+1}\ v_{j+2} \dots v_n\ v_1$$

obtained by deleting the edges v_iv_{i+1} and v_jv_{j+1} and adding the edges v_iv_j and $v_{i+1}v_{j+1}$, as shown in Figure 3.32. (If $j = n$ then C_{ij} is to be interpreted as $v_1v_2 \dots v_iv_nv_{n-1} \dots v_{i+1}v_1$.)

Now if the sum of the weights of the two new edges is less than that of the edges they replaced this new circuit C_{ij} is an improvement on C, i.e., if

$$w(v_iv_j) + w(v_{i+1}v_{j+1}) < w(v_iv_{i+1}) + w(v_jw_{j+1})$$

then C_{ij} is of smaller length than C. In this case we replace C by C_{ij} and perform a similar comparison on C_{ij}. We repeat the procedure until we reach a cycle which

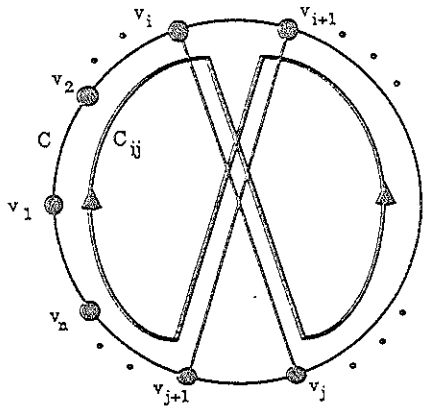


Figure 3.32: The Hamiltonian cycle C_{ij} obtained from C.

cannot be improved upon by using the same technique. This process is called the *two-optimal* method since at each stage we are choosing the optimal from two pairs of edges.

We carry out the procedure systematically, first by taking $i = 1$ and looking at the j values $3, 4, \dots, n$ in turn, then taking $i = 2$ and looking at the j values $4, 5, \dots, n$ in turn, and so on until we reach $i = n-2$, the final step having $i = n-2$, $j = n$. This leads us to the following two-optimal algorithm:

The Two-Optimal Algorithm

Step 1. Let $C = v_1\ v_2 \dots v_n\ v_1$ be any Hamiltonian cycle of the weighted graph G and let w be the weight of C, i.e.,

$$w = w(v_1v_2) + w(v_2v_3) + \dots + w(v_{n-1}v_n) + w(v_nv_1).$$

Step 2. Set $i = 1$.

Step 3. Set $j = i + 2$.

Step 4. Let C_{ij} be the Hamiltonian cycle

$$C_{ij} = v_1\ v_2 \dots v_i\ v_j\ v_{j-1} \dots v_{i+1}\ v_{j+1}\ v_{j+2} \dots v_n\ v_1$$

and let w_{ij} denote the weight of C_{ij}, so that

$$w_{ij} = w - w(v_iv_{i+1}) - w(v_jv_{j+1}) + w(v_iv_j) + w(v_{i+1}v_{j+1}).$$

If $w_{ij} < w$, i.e., if $w(v_iv_j)+w(v_{i+1}v_{j+1}) < w(v_iv_{i+1})+w(v_jw_{j+1})$, then replace C by C_{ij} and w by w_{ij}, i.e., set $C = C_{ij}, w = w_{ij}$, and return to Step 1, taking the sequence of vertices $v_1v_2 \dots v_nv_1$ as given by our new C.

Step 5. Set $j = j + 1$. If $j \leq n$, do Step 4. Otherwise set $i = i + 1$. If $i \leq n - 2$, do Step 3. Otherwise stop.

We illustrate the algorithm using the complete weighted graph G on $n = 6$ vertices shown in Figure 3.33.

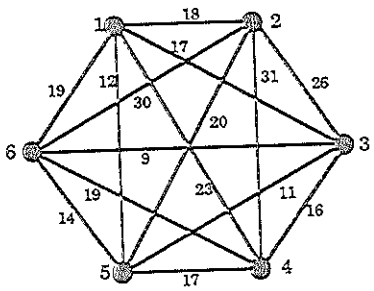


Figure 3.33

Step 1. Let $C = v_1\, v_2\, v_3\, v_4\, v_5\, v_6\, v_1$ be 1 2 3 4 5 6 1, so that
$$w = 18 + 26 + 16 + 17 + 14 + 19 = 110.$$

Step 2. Set $i = 1$.

Step 3. Set $j = i + 2 = 1 + 2 = 3$.

Step 4. Set $C_{13} = v_1\, v_3\, v_2\, v_4\, v_5\, v_6\, v_1 = 1\,3\,2\,4\,5\,6\,1$, so that
$$w_{13} = \underline{17} + 26 + \underline{31} + 17 + 14 + 19 = 124.$$
(Here, and in the following, the weights underlined are those of the two new edges.) Since $w_{13} > w$, we move on to

Step 5. Increase j to 4.

Step 4. Set $C_{14} = v_1\, v_4\, v_3\, v_2\, v_5\, v_6\, v_1 = 1\,4\,3\,2\,5\,6\,1$, so that
$$w14 = \underline{23} + 16 + 26 + \underline{20} + 14 + 19 = 118.$$
Since $w_{14} > w$, we move on to

Step 5. Increase j to 5.

Step 4. Set $C_{15} = v_1\, v_5\, v_4\, v_3\, v_2\, v_6\, v_1 = 1\,5\,4\,3\,2\,6\,1$, so that
$$w_{15} = \underline{12} + 17 + 16 + 26 + \underline{30} + 19 = 120.$$
Since $w_{15} > w$, we move on to

Step 5. Increase j to 6.

Step 4. Set $C_{16} = v_1\, v_6\, v_5\, v_4\, v_3\, v_2\, v_1 = 1\,6\,5\,4\,3\,2\,1$; this is just the reverse of the cycle C so that $w_{16} = 80$. Since $w_{16} = w$, we move on to

Step 5. Increase j to 7. Since $j > n$, we increase i to 2. Since $i \leq 4 = n - 2$, we move on to

Step 3. Set $j = i + 2 = 2 + 2 = 4$.

Step 4. Set $C_{24} = v_1\, v_2\, v_4\, v_3\, v_5\, v_6\, v_1 = 1\,2\,4\,3\,5\,6\,1$, so that
$$w_{24} = 18 + \underline{31} + 16 + \underline{11} + 14 + 19 = 109.$$
Since $w_{24} < w$, we replace C by C_{24} and w by 109 and move on to

Step 1. Now $C = 1\,2\,4\,3\,5\,6\,1$, as shown in Figure 3.34, and $w = 109$. We denote C by $v_1\, v_2\, v_3\, v_4\, v_5\, v_6\, v_1$, (not the same v's as before).

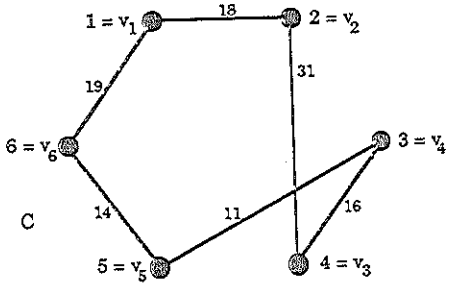


Figure 3.34: The new improved Hamiltonian cycle C.

Step 2. Set $i = 1$.

Step 3. Set $j = i + 2 = 1 + 2 = 3$.

Step 4. Set $C_{13} = v_1\, v_3\, v_2\, v_4\, v_5\, v_6\, v_1 = 1\,4\,2\,3\,5\,6\,1$, so that
$$w_{13} = \underline{23} + 31 + \underline{26} + 11 + 14 + 19 = 124.$$
Since $w_{13} > w$, we move on to

Step 5. Increase j to 4.

Step 4. Set $C_{14} = v_1\, v_4\, v_3\, v_2\, v_5\, v_6\, v_1 = 1\,3\,4\,2\,5\,6\,1$, so that
$$w_{14} = \underline{17} + 16 + 31 + \underline{20} + 14 + 19 = 117.$$
Since $w_{14} > w$, we move on to

Step 5. Increase j to 5.

Step 4. Set $C_{15} = v_1\, v_5\, v_4\, v_3\, v_2\, v_6\, v_1 = 1\,5\,3\,4\,2\,6\,1$, so that
$$w_{15} = \underline{12} + 11 + 16 + 31 + \underline{30} + 19 = 119.$$
Since $w_{15} > w$, we move on to

Step 5. Increase j to 6.

Step 4. Set $C_{16} = v_1\, v_6\, v_5\, v_4\, v_3\, v_2\, v_1 = 1\,6\,5\,3\,4\,2\,1$; this is just the reverse of the cycle C so that $w_{16} = 79$. Since $w_{16} = w$, we move on to

Step 5. Increase j to 7. Since $j > 6 = n$, we increase i to 2. Since $i \leq 4 = n-2$, we move on to

Step 3. Set $j = i+2 = 2+2 = 4$.

Step 4. Set $C_{24} = v_1\, v_2\, v_4\, v_3\, v_5\, v_6\, v_1 = 1\,2\,3\,4\,5\,6\,1$, so that
$$w_{24} = 18 + \underline{26} + 16 + \underline{17} + 14 + 19 = 110.$$
Since $w_{24} > w$, we move on to

Step 5. Increase j to 5.

Step 4. Set $C_{25} = v_1\, v_2\, v_5\, v_4\, v_3\, v_6\, v_1 = 1\,2\,5\,3\,4\,6\,1$, so that
$$w_{25} = 18 + \underline{20} + 11 + 16 + \underline{19} + 19 = 103.$$
Since $w_{25} < w$, we replace C by C_{25} and w by 103 and move on to

Step 1. Now $C = 1\,2\,5\,3\,4\,6\,1$, as shown in Figure 3.35, and $w = 103$. We denote C by $v_1\, v_2\, v_3\, v_4\, v_5\, v_6\, v_1$, (not the same v's as before).

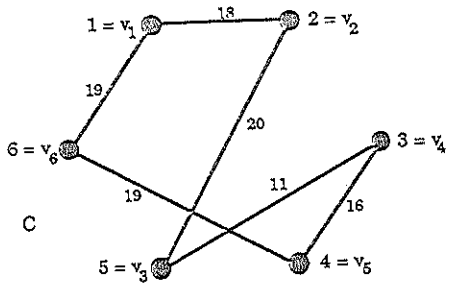


Figure 3.35: The new improved Hamiltonian cycle C.

Set $i = 1$.

Set $j = i+2 = 1+2 = 3$.

Set $C_{13} = v_1\, v_3\, v_2\, v_4\, v_5\, v_6\, v_1 = 1\,5\,2\,3\,4\,6\,1$, so that
$$w_{13} = \underline{12} + 20 + \underline{26} + 16 + 19 + 19 = 112.$$

We omit most of the remaining details. The reader can check that the next change occurs with $i = 3$ and $j = 5$, so that

$$C_{ij} = C_{35} = v_1\, v_2\, v_3\, v_4\, v_6\, v_1 = 1\,2\,5\,4\,3\,6\,1$$

and $w_{ij} = w_{35} = 18 + 20 + 17 + 16 + 9 + 19 = 99$, giving the new cycle C as shown in Figure 3.36.

Moreover if we continue the process, no further improvement is obtained.

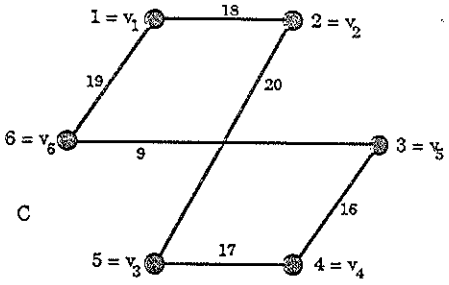


Figure 3.36: The optimal Hamiltonian cycle C.

We now indicate how to get some idea of how good this solution is. Given any optimal circuit C in G, then for any vertex v of G, $C - v$ will be a spanning tree of $G - v$. Now if T is an optimal spanning tree of $G - v$ and if e and f are two edges incident with v such that $w(e) + w(f)$ is as small as possible then certainly $w(T) + w(e) + w(f)$ is not greater than $w(C)$ i.e. $w(T) + w(e) + w(f) < w(C)$. Now Kruskal's algorithm gives us a spanning tree and applying it to our graph $G - v$ with v as vertex 2 we get the following optimal tree T.

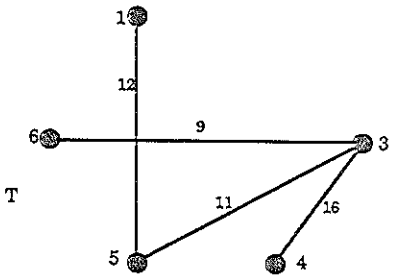


Figure 3.37: A minimal spanning tree T, of weight 48, for the graph $G - 2$.

On examination, the two edges incident with vertex 2 with smallest weights are those joining 2 to 1 and 5 with weights 13 and 15 respectively. Thus these are chosen as our e and f and we get

$$w(T) + w(e) + w(f) = 48 + 18 + 20 = 86.$$

Hence any optimal cycle C has weight $w(C) > 86$. So our solution, which has weight 99, seems reasonable — note that it may indeed be the best possible.

We now consider a different method, again finding a reasonably good solution, but with no guarantee that it is the best. It is called the *closest insertion algorithm*. The general idea of this method is to gradually build up a sequence of cycles in the graph which involve more and more vertices, until all the vertices are used up, and to

involve one more vertex at each stage by determining which vertex, as yet unchosen, is nearest to the cycle already created and then inserting this vertex into the cycle in as economical way as possible.

We will now be more specific, using an example to illustrate the method.

In the Real Cool Ice Cream Company's factory, six different flavours of ice cream are produced in sequence on one machine. The machine must be cleaned after each flavour before the production of the next flavour and the cleaning time depends on the two flavours (but we will assume that it does not matter which of the two flavours is mixed first). Real Cool wishes to find an ordering of the six flavours, starting with banana, so that, if the machine produces the six flavours (once and) only once in this order, then the total cleaning time spent is smallest possible.

The cleaning times (in minutes) are given in the following table:

flavour	banana	chocolate	mint	raspberry	strawberry	vanilla
banana	0	10	11	11	6	10
chocolate	10	0	17	15	15	20
mint	11	17	0	10	15	19
raspberry	11	15	10	0	15	20
strawberry	6	15	15	15	0	11
vanilla	10	20	19	20	11	0

This table produces the complete weighted graph of Figure 3.38, where B denotes "banana", C denotes "chocolate", etc.

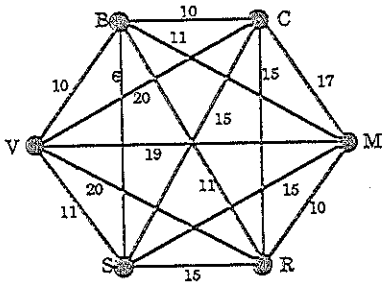


Figure 3.38

We now describe the closest insertion algorithm. The description uses the idea of the distance of a vertex v from a walk W. This is defined to be

$$d(v, W) = \min\{d(v, u) : u \text{ is a vertex of } W\}.$$

The vertex v, not in W, is then said to be closest to W if $d(v, W) \leq d(x, W)$ for any other vertex x not in W.

We emphasise the way in which a new vertex is inserted into a cycle by using bold type.

The Closest Insertion Algorithm

Step 1. Choose any vertex v_1 as a starting vertex.

Step 2. From among the $n-1$ vertices not chosen so far, find one, say v_2, which is closest to v_1. Let W_2 denote the walk $v_1 v_2 v_1$.

Step 3. From among the $n-2$ vertices not chosen so far, find one, say v_3, which is closest to the walk $W_2 = v_1\, v_2\, v_1$.
Let W_3 be the walk $v_1 v_2 v_3 v_1$ (so that W_3 is in fact a cycle).

Step 4. From among the $n-3$ vertices not chosen so far, find one, say v_4, which is closest to the walk W_3.
Determine which of the walks (cycles) $v_1\, v_2\, v_3\, v_4\, v_1$, $v_1\, v_2\, v_4\, v_3\, v_1$, $v_1\, v_4\, v_2\, v_3\, v_1$ is the shortest.
Let W_4 denote the shortest one and relabel it, if necessary, as $v_1\, v_2\, v_3\, v_4\, v_1$.

Step 5. From among the $n-4$ vertices not chosen so far, find one, say v_5 which is closest to the walk W_4.
Determine which of the cycles $v_1\, v_2\, v_3\, v_4\, v_5\, v_1$, $v_1\, v_2\, v_3\, v_5\, v_4\, v_1$, $v_1\, v_2\, v_5\, v_3\, v_4\, v_1$, $v_1\, v_5\, v_2\, v_3\, v_4\, v_1$ is shortest.
Let W_5 denote the shortest one and relabel it, if necessary , as $v_1\, v_2\, v_3\, v_4\, v_5\, v_1$.
Continue in this way to eventually arrive at

Step n-1. From the 2 vertices not already chosen, find one, say v_{n-1} which is closest to the walk (cycle) $W_{n-2} = v_1\, v_2 \ldots v_{n-2}\, v_1$.
Determine which of the cycles $v_1\, v_2 \ldots v_{n-2}\, v_{n-1}\, v_1$, $v_1\, v_2 \ldots v_{n-3}\, v_{n-1}\, v_{n-2}\, v_1$, $\ldots$, $v_1\, v_{n-1}\, v_2\, v_3 \ldots v_{n-2}\, v_1$ is shortest.
Let W_{n-1} denote the shortest of these cycles and relabel it, if necessary, as $v_1\, v_2 \ldots v_{n-2}\, v_{n-1}\, v_1$.

Step n. Denote the remaining unchosen vertex by v_n and determine which of the cycles $v_1\, v_2 \ldots v_{n-1}\, v_n\, v_1$, $v_1\, v_2 \ldots v_{n-2}\, v_n\, v_{n-1}\, v_1$, $\ldots$, $v_1\, v_n\, v_2\, v_3 \ldots v_{n-1}\, v_1$ is shortest.
Let W_n denote the shortest of these cycles.

Conclusion: W_n is a Hamiltonian cycle of G which is, at least approximately, optimal.

For our example above we have

Step 1. Let $v_1 = B$.

Step 2. $v_2 = S$ is closest to B so $W_2 = v_1\, v_2\, v_1 = BSB$.

Step 3. $v_3 = V$ is closest to W_2. (It has distance 10 from B; but there is actually a choice here since C is another candidate.)
W_3 is the cycle $v_1\, v_2\, v_3\, v_1 = BSVB$.

Step 4. $v_4 = C$ is closest to W_3 (It has distance 10 from B.)
We find the lengths of the cycles $v_1\,v_2\,v_3\,v_4\,v_1$, $v_1\,v_2\,v_4\,v_3\,v_1$ and $v_1\,v_4\,v_3\,v_2\,v_1$:
$v_1\,v_2\,v_3\,v_4\,v_1 = BSVCB$ has length $6+11+20+10=47$,
$v_1\,v_2\,v_4\,v_3\,v_1 = BSCVB$ has length $6+15+20+10=51$,
$v_1\,v_4\,v_2\,v_3\,v_1 = BCSVB$ has length $10+15+11+10=46$.
We let W_4 denote the shortest of these, namely $BCSVB$, and relabel with $BCSVB = v_1\,v_2\,v_3\,v_4\,v_1$.

Step 5. $v_5 = R$ is closest to W_4. (It has distance 11 from B; but again there is a choice here: M is also distance 11 from W_4.)
We find the lengths of the cycles $v_1\,v_2\,v_3\,v_4\,v_5\,v_1$, $v_1\,v_2\,v_3\,v_5\,v_4\,v_1$, $v_1\,v_2\,v_5\,v_3\,v_4\,v_1$ and $v_1\,v_5\,v_2\,v_3\,v_4\,v_1$:
$v_1\,v_2\,v_3\,v_4\,v_5\,v_1 = BCSVRB$ has length $10+15+11+20+11=67$,
$v_1\,v_2\,v_3\,v_5\,v_4\,v_1 = BCSRVB$ has length $10+15+15+20+10=70$,
$v_1\,v_2\,v_5\,v_3\,v_4\,v_1 = BCRSVB$ has length $10+15+15+11+10=61$,
$v_1\,v_5\,v_2\,v_3\,v_4\,v_1 = BRCSVB$ has length $11+15+15+11+10=62$.
We let W_5 denote the shortest of these, namely $BCRSVB$, and relabel with $BCRSVB = v_1\,v_2\,v_3\,v_4\,v_5\,v_1$.

Step n ($n=6$). Let $v_6 = M$, the last remaining vertex.
We find the lengths of the cycles $v_1\,v_2\,v_3\,v_4\,v_5\,v_6\,v_1$, $v_1\,v_2\,v_3\,v_4\,v_6\,v_5\,v_1$, $v_1\,v_2\,v_3\,v_6\,v_4\,v_5\,v_1$, $v_1\,v_2\,v_6\,v_3\,v_4\,v_5\,v_1$ and $v_1\,v_6\,v_2\,v_3\,v_4\,v_5\,v_1$:
$v_1\,v_2\,v_3\,v_4\,v_5\,v_6\,v_1 = BCRSVMB$ has length $10+15+15+11+19+11=81$,
$v_1\,v_2\,v_3\,v_4\,v_6\,v_5\,v_1 = BCRSMVB$ has length $10+15+15+15+19+10=84$,
$v_1\,v_2\,v_3\,v_6\,v_4\,v_5\,v_1 = BCRMSVB$ has length $10+15+10+15+11+10=71$,
$v_1\,v_2\,v_6\,v_3\,v_4\,v_5\,v_1 = BCMRSVB$ has length $10+17+10+15+11+10=73$,
$v_1\,v_6\,v_2\,v_3\,v_4\,v_5\,v_1 = BMCRSVB$ has length $11+17+15+15+11+10=79$.

Then the shortest cycle is $W_6 = BCRMSVB$ and it has length 71.

We can therefore conclude that a reasonably efficient cleaning-time cycle for Real Cool's ice cream machine is
banana-chocolate-raspberry-mint-strawberry-vanilla-banana,
the cycle taking 71 minutes to complete.

Exercises for Section 3.4

3.4.1 Six computer programs, $P_1, \ldots, P_6$, have to be run in sequence on a mainframe machine. Each program needs its own resources such as a part of the main memory, a compiler and drives, and changing from one set of resources to another uses up valuable time. The following matrix $C = (c_{ij})$ records the times c_{ij} used in converting from the resources for program P_i to those for program P_j.

$$C = \begin{bmatrix} 0 & 30 & 90 & 10 & 30 & 10 \\ 30 & 0 & 80 & 40 & 20 & 20 \\ 90 & 80 & 0 & 40 & 20 & 30 \\ 10 & 40 & 40 & 0 & 60 & 10 \\ 30 & 20 & 20 & 60 & 0 & 90 \\ 10 & 20 & 30 & 10 & 90 & 0 \end{bmatrix}.$$

Note that $c_{ij} = c_{ji}$ for each i and j. Using both the two optimal method and the closest insertion method for the Travelling Salesman Problem, find an ordering of the programs which, if they are run in this order, should result in a comparatively small total conversion time.

3.4.2 Carry out the two optimal method and the closest insertion method for the Travelling Salesman Problem for the complete weighted graph of Figure 3.39.

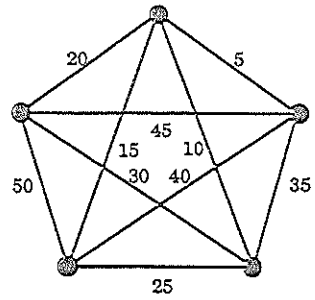


Figure 3.39