6.2. Paths and Cycles

6.2.1. Paths. A *path* from v_0 to v_n of length n is a sequence of $n+1$ vertices (v_k) and n edges (e_k) of the form $v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n$, where each edge e_k connects v_{k-1} with v_k. If there are no parallel edges we only need to specify the vertices: $v_0, v_1, v_2, \ldots, v_n$.

A *simple path* from v to w is a path from v to w with no repeated vertices. A *cycle* (or *circuit*) is a path of non-zero length from v to v with no repeated edges. A *simple cycle* is a *cycle* with no repeated vertices (except for the beginning and ending vertex).

Remark: If a graph contains a cycle from v to v, then it contains a simple cycle from v to v. Proof: if a given vertex v_i occurs twice in the cycle, we can remove the part of it that goes from v_i and back to v_i. If the resulting cycle still contains repeated vertices we can repeat the operation until there are no more repeated vertices.

6.2.2. Connected Graphs. A graph G is called *connected* if there is a path between any two distinct vertices of G. Otherwise the graph is called *disconnected*. A directed graph is connected if its associated undirected graph (obtained by ignoring the directions of the edges) is connected. A *connected component* of G is any connected subgraph $G' = (V', E')$ of $G = (V, E)$ such that there is not edge (in G) from a vertex in V to a vertex in $V - V'$. Given a vertex in G, the component of G containing v is the subgraph G' of G consisting of all edges and vertices of g contained in some path beginning at v.

6.2.3. The Seven Bridges of Königsberg. This is a classical problem that started the discipline today called *graph theory*.

During the eighteenth century the city of Königsberg (in East Prussia) was divided into four sections, including the island of Kneiphop, by the Pregel river. Seven bridges connected the regions, as shown in figure 6.8. It was said that residents spent their Sunday walks trying to find a way to walk about the city so as to cross each bridge exactly once and then return to the starting point. The first person to solve the problem (in the negative) was the Swiss mathematician Leonhard Euler in 1736. He represented the sections of the city and the seven bridges by the graph of figure 6.9, and proved that it is impossible to find a path in it that transverses every edge of the graph exactly once. In the next section we study why this is so.

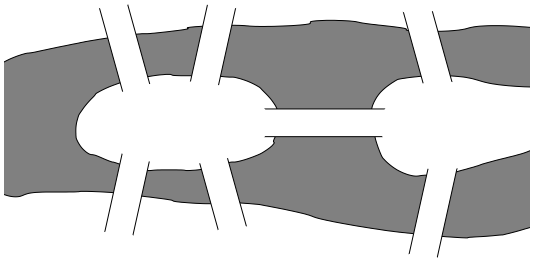

FIGURE 6.8. The Seven Bridges of Königsberg.

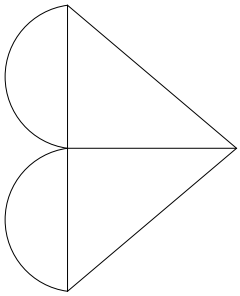

FIGURE 6.9. Graph for the Seven Bridges of Königsberg.

6.2.4. Euler paths and cycles. Let $G = (V, E)$ be a graph with no isolated vertices. An *Euler path* in G is a path that transverses every edge of the graph exactly once. Analogously, an *Euler cycle* in G is a cycle that transverses every edge of the graph exactly once.

The graphs that have an Euler path can be characterized by looking at the degree of their vertices. Recall that the degree of a vertex v, represented $\delta(v)$, is the number of edges that contain v (loops are counted twice). An *even* vertex is a vertex with even degree; an *odd* vertex is a vertex with odd degree. The sum of the degrees of all vertices in a graph equals twice its number of edges, so it is an even number. As a consequence, the number of odd vertices in a graph is always even.

Then G contains an Euler cycle if and only if G is connected and all its vertices have even degree. Also, G contains an Euler path from vertex a to vertex b ($\neq a$) if and only if G is connected, a and b have odd degree, and all its other vertices have even degree.

6.2.5. Hamiltonian Cycles. A *Hamiltonian cycle* in a graph G is a cycle that contains each vertex of G once (except for the starting

and ending vertex, which occurs twice). A *Hamiltonian path* in G is a path (not a cycle) that contains each vertex of G once. Note that by deleting an edge in a Hamiltonian cycle we get a Hamilton path, so if a graph has a Hamiltonian cycle, then it also has a Hamiltonian path. The converse is not true, i.e., a graph may have a Hamiltonian path but not a Hamiltonian cycle. *Exercise*: Find a graph with a Hamiltonian path but no Hamiltonian cycle.

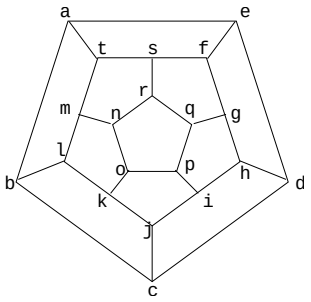

FIGURE 6.10. Hamilton's puzzle.

In general it is not easy to determine if a given graph has a Hamiltonian path or cycle, although often it is possible to argue that a graph has no Hamiltonian cycle. For instance if $G = (V, E)$ is a bipartite graph with vertex partition $\{V_1, V_2\}$ (so that each edge in G connects some vertex in V_1 to some vertex in V_2), then G cannot have a Hamiltonian cycle if $|V_1| \neq |V_2|$, because any path must contain alternatively vertices from V_1 and V_2, so any cycle in G must have the same number of vertices from each of both sets.

Edge removal argument. Another kind of argument consists of removing edges trying to make the degree of every vertex equal two. For instance in the graph of figure 6.11 we cannot remove any edge because that would make the degree of b, e or d less than 2, so it is impossible to reduce the degree of a and c. Consequently that graph has no Hamiltonian cycle.

The Traveling Salesperson Problem. Given a weighted graph, the *traveling salesperson problem* (TSP) consists of finding a Hamiltonian cycle of minimum length in this graph. The name comes from a classical problem in which a salesperson must visit a number of cities and go back home traveling the minimum distance possible. One way to solve the problem consists of searching all possible Hamiltonian cycles and computing their length, but this is very inefficient. Unfortunately no

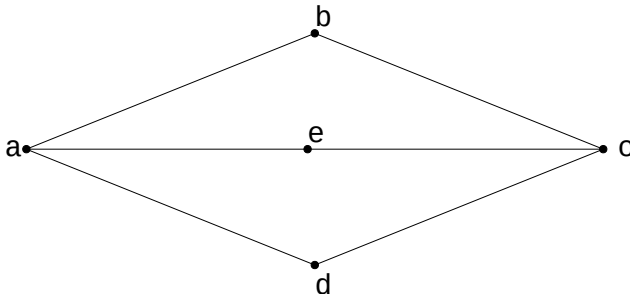

FIGURE 6.11. Graph without Hamiltonian cycle.

efficient algorithm is known for solving this problem (and chances are that none exists).

Remark: (Digression on P/NP problems.) Given a weighted graph with n vertices the problem of determining whether it contains a Hamiltonian cycle of length not greater than a given L is known to be NP-complete. This means the following. First it is a *decision* problem, i.e., a problem whose solution is "yes" or "no". A decision problem is said to be polynomial, or belong to the class P, if it can be solved with an algorithm of complexity $O(n^k)$ for some integer k. It is said to be non-deterministic polynomial, or belong to the class NP, if in all cases when the answer is "yes" this can be determined with a non-deterministic algorithm of complexity $O(n^k)$. A non-deterministic algorithm is an algorithm that works with an extra hint, for instance in the TSP, if G has a Hamiltonian cycle of length not greater than L the hint could consist of a Hamiltonian cycle with length not greater than L—so the task of the algorithm would be just to check that in fact that length is not greater than L.[1] Currently it is not known whether the class NP is strictly larger than the class P, although it is strongly suspected that it is. The class NP contains a subclass called NP-complete containing the "hardest" problems of the class, so that their complexity must be higher than polynomial unless P=NP. The TSP is one of these problems.

Gray Codes. A *Gray code* is a sequence $s_1, s_2, \ldots, s_{2^n}$ of n-binary strings verifying the following conditions:

[1]Informally, P problems are "easy to solve", and NP problems are problems whose answer is "easy to check". In a sense the P=NP problem consist of determining whether every problem whose solution is easy to check is also easy to solve.

1. Every n-binary string appears somewhere in the sequence.
2. Two consecutive strings s_i and s_{i+1} differ exactly in one bit.
3. s_{2^n} and s_1 differ in exactly one bit.

For instance: $000, 001, 011, 010, 110, 111, 101, 100,$

The problem of finding a gray code is equivalent to finding a Hamiltonian cycle in the n-cube.

6.2.6. Dijkstra's Shortest-Path Algorithm. This is an algorithm to find the shortest path from a vertex a to another vertex z in a connected weighted graph. Edge (i, j) has weight $w(i, j) > 0$, and vertex x is labeled $L(x)$ (minimum distance from a if known, otherwise ∞). The output is $L(z) =$ length of a minimum path from a to z.

```
 1: procedure dijkstra(w,a,z,L)
 2:   L(a) := 0
 3:   for all vertices x ≠ a do
 4:     L(x) := ∞
 5:   T := set of all vertices
      // T is the set of all vertices whose shortest
      // distance from a has not been found yet
 6:   while z in T do
 7:     begin
 8:       choose v in T with minimum L(v)
 9:       T := T - {v}
10:       for each x in T adjacent to v do
11:         L(x) := min{L(x),L(v)+w(v,x)}
12:     end
13:   return(L(z))
14: end dijkstra
```

For instance consider the graph in figure 6.12.

The algorithm would label the vertices in the following way in each iteration (the boxed vertices are the ones removed from T):

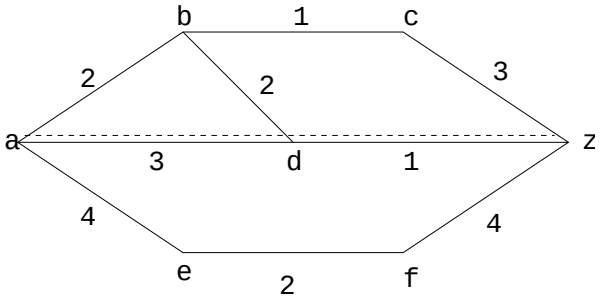

FIGURE 6.12. Shortest path from a to z.

iteration	a	b	c	d	e	f	z
0	0	∞	∞	∞	∞	∞	∞
1	$\boxed{0}$	2	∞	3	4	∞	∞
2	$\boxed{0}$	$\boxed{2}$	3	3	4	∞	∞
3	$\boxed{0}$	$\boxed{2}$	$\boxed{3}$	3	4	∞	6
4	$\boxed{0}$	$\boxed{2}$	$\boxed{3}$	$\boxed{3}$	4	∞	4
5	$\boxed{0}$	$\boxed{2}$	$\boxed{3}$	$\boxed{3}$	$\boxed{4}$	6	4
6	$\boxed{0}$	$\boxed{2}$	$\boxed{3}$	$\boxed{3}$	$\boxed{4}$	6	$\boxed{4}$

At this point the algorithm returns the value 4.

Complexity of Dijkstra's algorithm. For an n-vertex, simple, connected weighted graph, Dijkstra's algorithm has a worst-case run time of $\Theta(n^2)$.