## 7.3. Paths and Circuits

**7.3.1.** 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  $v_k$  (and points from  $v_{k-1}$  to  $v_k$  if the edge is directed). The path may be specified by giving only the sequence of edges  $e_1, \ldots, e_n$ . If there are no multiple edges we can specify the path by giving only the vertices:  $v_0, v_1, \ldots, v_n$ . The path is a circuit (or cycle) if it begins and ends at the same vertex, i.e.,  $v_0 = v_n$ , and has length greater than zero. A path or circuit is simple if it does not contain the same edge twice.

**7.3.2.** 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 directed graph is *strongly connected* if for every pair of distict points u, v, there is a path from u to v and there is a path from v to u. 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.

**7.3.3.** 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)<sup>1</sup> was divided into four sections, including the island of Kneiphop, by the Pregel river. Seven bridges connected the regions, as shown in figure 7.11. 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 7.12, 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.

 $<sup>^{1}\</sup>mathrm{The}$  city is currently called Kaliningrad and is part of the Russian republic.

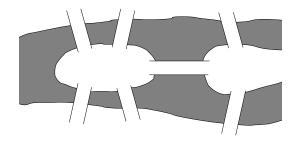


FIGURE 7.11. The Seven Bridges of Königsberg.

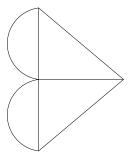


FIGURE 7.12. Graph for the Seven Bridges of Königsberg.

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

Existence of Euler Paths and Circuits. 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  $\deg(v)$ , 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.

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

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

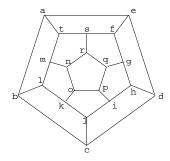


FIGURE 7.13. Hamilton's puzzle.

In general it is not easy to determine if a given graph has a Hamilton path or circuit, although often it is possible to argue that a graph has no Hamilton circuit. 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 Hamilton circuit if  $|V_1| \neq |V_2|$ , because any path must contain alternatively vertices from  $V_1$  and  $V_2$ , so any circuit 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 7.14 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 Hamilton circuit.

Dirac's Theorem. If G is a simple graph with n vertices with  $n \geq 3$  such that the degree of every vertex in G is at least n/2, then G has a Hamilton circuit.

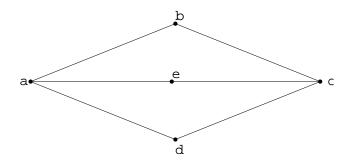


FIGURE 7.14. Graph without Hamilton circuit.

Ore's Theorem. If G is a simple graph with n vertices with  $n \geq 3$  such that  $\deg(u) + \deg(v) \geq n$  for every pair of nonadjacent vertices u and v in G, then G has a Hamilton circuit.

The Traveling Salesperson Problem. Given a weighted graph, the traveling salesperson problem (TSP) consists of finding a Hamilton circuit 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 Hamilton circuits and computing their length, but this is very inefficient. Unfortunately no 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 Hamilton circuit 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 Hamilton circuit of length not greater than L the hint could consist of a Hamilton circuit 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.<sup>2</sup> 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. 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 Hamilton circuit in the *n*-cube.

**7.3.6.** 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  $\infty$ ). The output is L(z) = length of a minimum path from a to z.

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

<sup>&</sup>lt;sup>2</sup>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.

14: end

15: return L(z)16: end dijkstra

For instance consider the graph in figure 7.15.

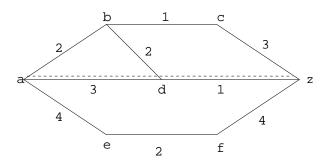


FIGURE 7.15. Shortest path from a to z.

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

| iteration | a | b        | c        | d        | e        | f        | z        |
|-----------|---|----------|----------|----------|----------|----------|----------|
| 0         | 0 | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 1         | 0 | 2        | $\infty$ | 3        | 4        | $\infty$ | $\infty$ |
| 2         | 0 | 2        | 3        | 3        | 4        | $\infty$ | $\infty$ |
| 3         | 0 | 2        | 3        | 3        | 4        | $\infty$ | 6        |
| 4         | 0 | 2        | 3        | 3        | 4        | $\infty$ | 4        |
| 5         | 0 | 2        | 3        | 3        | 4        | 6        | 4        |
| 6         | 0 | 2        | 3        | 3        | 4        | 6        | 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  $\Theta(n^2)$ .