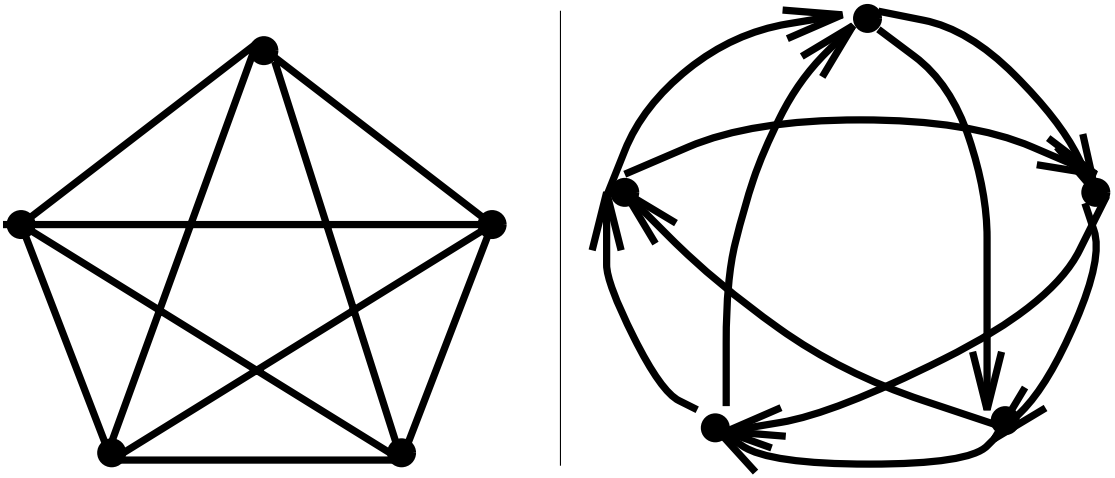

INPUT OUTPUT

15.7 Eulerian Cycle/Chinese Postman

Input description: A graph $G = (V, E)$.

Problem description: Find the shortest tour visiting each edge of G at least once.

Discussion: Suppose you are given the map of a city and charged with designing the routes for garbage trucks, snow plows, or postmen. In each of these applications, every road in the city must be completely traversed at least once in order to ensure that all deliveries or pickups are made. For efficiency, you seek to minimize total drive time, or (equivalently) the total distance or number of edges traversed.

Alternately, consider a human-factors validation of telephone menu systems. Each "Press 4 for more information" option is properly interpreted as an edge between two vertices in a graph. Our tester seeks the most efficient way to walk over this graph and visit every link in the system at least once.

Such applications are variants of the *Eulerian cycle* problem, best characterized by the puzzle that asks children to draw a given figure completely without (1) without repeating any edges, or (2) lifting their pencil off the paper. They seek a path or cycle through a graph that visits each edge exactly once.

Well-known conditions exist for determining whether a graph contains an Eulerian cycle or path:

- An *undirected* graph contains an Eulerian *cycle* iff (1) it is connected, and (2) each vertex is of even degree.

- An *undirected* graph contains an Eulerian *path* iff (1) it is connected, and (2) all but two vertices are of even degree. These two vertices will be the start and end points of any path.

- A *directed* graph contains an Eulerian *cycle* iff (1) it is strongly-connected, and (2) each vertex has the same in-degree as out-degree.

- Finally, a *directed* graph contains an Eulerian *path* from x to y iff (1) it is connected, and (2) all other vertices have the same in-degree as out-degree, with x and y being vertices with in-degree one less and one more than their out-degrees, respectively.

This characterization of Eulerian graphs makes it is easy to test whether such a cycle exists: verify that the graph is connected using DFS or BFS, and then count the number of odd-degree vertices. Explicitly constructing the cycle also takes linear time. Use DFS to find an arbitrary cycle in the graph. Delete this cycle and repeat until the entire set of edges has been partitioned into a set of edge-disjoint cycles. Since deleting a cycle reduces each vertex degree by an even number, the remaining graph will continue to satisfy the same Eulerian degree-bound conditions. These cycles will have common vertices (since the graph is connected), and so can be spliced together in a "figure eight" at a shared vertex. By so splicing all the extracted cycles together, we construct a single circuit containing all of the edges.

An Eulerian cycle, if one exists, solves the motivating snowplow problem, since any tour that visits each edge only once must have minimum length. However, it is unlikely that your road network happens to satisfy the Eulerian degree conditions. Instead, we need to solve the more general *Chinese postman problem*, which minimizes the length of a cycle that traverses every edge at least once. This minimum cycle will never visit any edge more than twice, so good tours exist for any road network.

The optimal postman tour can be constructed by adding the appropriate edges to the graph G to make it Eulerian. Specifically, we find the shortest path between each pair of odd-degree vertices in G. Adding a path between two odd-degree vertices in G turns both of them to even-degree, moving G closer to becoming an Eulerian graph. Finding the best set of shortest paths to add to G reduces to identifying a minimum-weight perfect matching in a special graph G'.

For undirected graphs, the vertices of G' correspond the odd-degree vertices of G, with the weight of edge (i, j) defined to be the length of the shortest path from i to j in G. For directed graphs, the vertices of G' correspond to the degree-imbalanced vertices from G, with the bonus that all edges in G' go from out-degree deficient vertices to in-degree deficient ones. Thus, bipartite matching algorithms suffice when G is directed. Once the graph is Eulerian, the actual cycle can be extracted in linear time using the procedure just described.

Implementations: Several graph libraries provide implementations of Eulerian cycles, but Chinese postman implementations are rarer. We recommend the imple-

mentation of directed Chinese postman by Thimbleby [Thi03]. This Java implementation is available at *http://www.cs.swan.ac.uk/~csharold/cpp/index.html*.

GOBLIN (*http://www.math.uni-augsburg.de/~fremuth/goblin.html*) is an extensive C++ library dealing with all of the standard graph optimization problems, including Chinese postman for both directed and undirected graphs. LEDA (see Section 19.1.1 (page 658)) provides all the tools for an efficient implementation: Eulerian cycles, matching, and shortest-paths bipartite and general graphs.

Combinatorica [PS03] provides Mathematica implementations of Eulerian cycles and de Bruijn sequences. See Section 19.1.9 (page 661).

Notes: The history of graph theory began in 1736, when Euler [Eul36] first solved the seven bridges of Königsberg problem. Königsberg (now Kaliningrad) is a city on the banks of the Pregel river. In Euler's day there were seven bridges linking the banks and two islands, which can be modeled as a multigraph with seven edges and four vertices. Euler sought a way to walk over each of the bridges exactly once and return home—i.e. , an Eulerian cycle. Euler proved that such a tour is impossible, since all four of the vertices had odd degrees. The bridges were destroyed in World War II. See [BLW76] for a translation of Euler's original paper and a history of the problem.

Expositions on linear-time algorithms for constructing Eulerian cycles [Ebe88] include [Eve79a, Man89]. Fleury's algorithm [Luc91] is a direct and elegant approach to constructing Eulerian cycles. Start walking from any vertex, and erase any edge that has been traversed. The only criterion in picking the next edge is that we avoid using a bridge (an edge whose deletion disconnects the graph) until no other alternative remains.

The Euler's tour technique is an important paradigm in parallel graph algorithms. Many parallel graph algorithms start by finding a spanning tree and then rooting the tree, where the rooting is done using the Euler tour technique. See parallel algorithms texts (e.g., [JÓ2]) for an exposition, and [CB04] for recent experience in practice. Efficient algorithms exist to count the number of Eulerian cycles in a graph [HP73].

The problem of finding the shortest tour traversing all edges in a graph was introduced by Kwan [Kwa62], hence the name *Chinese* postman. The bipartite matching algorithm for solving Chinese postman is due to Edmonds and Johnson [EJ73]. This algorithm works for both directed and undirected graphs, although the problem is NP-complete for mixed graphs [Pap76a]. Mixed graphs contain both directed and undirected edges. Expositions of the Chinese postman algorithm include [Law76].

A *de Bruijn* sequence S of span n on an alphabet Σ of size α is a circular string of length α^n containing all strings of length n as substrings of S, each exactly once. For example, for $n = 3$ and $\Sigma = \{0, 1\}$, the circular string 00011101 contains the following substrings in order: 000, 001, 011, 111, 110, 101, 010, 100. De Bruijn sequences can be thought of as "safe cracker" sequences, describing the shortest sequence of dial turns with α positions sufficient to try out all combinations of length n.

De Bruijn sequences can be constructed by building a directed graph whose vertices are all α^{n-1} strings of length $n-1$, where there is an edge (u, v) iff $u = s_1 s_2 \ldots s_{n-1}$ and $v = s_2 \ldots s_{n-1} s_n$. Any Eulerian cycle on this graph describes a de Bruijn sequence. Expositions on de Bruijn sequences and their construction include [Eve79a, PS03].

Related Problems: Matching (see page 498), Hamiltonian cycle (see page 538).