times—an obvious source of inefficiency. An improvement suggested by Paton [11-48] reduces the relabeling of edges by the following devices: Instead of relabeling the edges in a fundamental circuit as soon as it is generated, we wait till a vertex z in Algorithm 3 has been completely examined, and then assign identical labels to the fundamental circuits (passing through z) thus generated. Therefore, labeling has to be performed only n times and not μ times. Moreover, we need not label every edge in the graph (Problem 11-11). It is left for the reader to construct a flow chart for the block-identification algorithm in which Algorithm 3 is completely embedded. Remaining details can be found in [11-48].

Using an entirely different approach, Hopcroft and Tarjan [11-31] and Tarjan [11-61] have given an algorithm which is faster than the algorithm described here for certain types of graphs. Their algorithm uses depth-first search on the graph (to be discussed later in this chapter), and the graph is to be input in the successor-listing form. Its execution time is proportional to e, whereas the time bound for the Read-Paton algorithm described here is proportional to n^{γ}, where $1 \leq \gamma \leq 2$, depending on the structure of the graph. Analysis and extensive tests show that for a typical graph of n vertices and e edges, Hopcroft and Tarjan's algorithm outperforms (on IBM 7044) Paton's algorithm as long as $e \leq 5n$. For graphs of much higher densities (Problem 11-1) Paton's algorithm performed better. Thus for planar graphs (since $e \leq 3n - 6$), Hopcroft and Tarjan's algorithm will in general be faster. Typically, for an 80-vertex 400-edge graph, the IBM 7044 took about 7 seconds for block identification with either of the two algorithms.

Algorithm 5: Directed Circuits

One of the most important things about a digraph is its directed circuits (also called cycles). The significance of directed circuits in many applications was discussed in Chapter 9. Unlike the case of undirected graphs, no technique is known by which we can obtain a basic set of directed circuits such that every directed circuit in the digraph is obtained as a linear combination of this basic set. Therefore, Algorithm 3 is of little help in obtaining all directed circuits of a digraph. We must generate every directed circuit individually. For this we must examine each edge (unless the edge is known a priori to belong to no directed circuit) many times.

Preliminary Simplification: Although it is not necessary, in most cases the prior application of the following two steps will simplify a given digraph. First, if the digraph is likely to be disconnected, use Algorithm 1 [with slight modification for a digraph (Problem 11-7)] to identify the connected components, and then consider one component at a time. Second, successively delete all vertices (and the edges incident on them) that have zero in-degree or zero out-degree. Clearly, such a vertex cannot lie in any directed circuit. These vertices are easy to identify because they correspond to entire rows

[for $d^+(v) = 0$] or columns [for $d^-(v) = 0$] of zeros in the adjacency matrix X. For example, if the digraph given was the one in Fig. 9-16, edges a, b, and h would be eliminated. Then in the next go-round edges e and c would be deleted, leaving us a digraph of only three edges, d, f, and g. On the other hand, this method of simplification will not reduce the digraph shown in Fig. 9-21.

Description of the Algorithm: This algorithm, first proposed by Roberts and Flores [11-56] and subsequently systematized by Tiernan [11-63], uses an exhaustive search to find all directed circuits in a given digraph G. As usual, the vertices of G are assigned integers $1, 2, \ldots, n$ as their names. The algorithm depends on starting from a vertex p_1 and building a directed path $P = (p_1, p_2, \ldots, p_k)$ until no further vertices (satisfying certain conditions) are "available" at vertex p_k. At p_k, when it is not possible to extend the directed path any further, the algorithm checks to see if there is a directed edge from p_k to p_1. If there is such an edge, a directed circuit $(p_1, p_2, \ldots, p_k, p_1)$ has been found and is duly recorded. If there is no such edge in the digraphs, we move back one vertex to p_{k-1} and try extending the path again from p_{k-1} along a different edge (if there is one). Whether a directed circuit is found or not, the algorithm makes vertex p_k forbidden for the next extension from p_{k-1} (thus avoiding going over the same path).

This process of looking for directed circuits and then moving back a vertex is continued till we finally backtrack to the vertex p_1 itself. Thus all directed paths starting from p_1 have been examined and directed circuits recorded. Starting with the next vertex, the entire process is repeated. The iteration starts with vertex $p_1 = 1$ and ends with $p_1 = n$.

In this exhaustive search for directed paths we must take the following precautions:

1. In the process of extending each directed path, going round and round a directed circuit must be avoided. This is achieved by insisting that any vertex that has already been included in the directed path is "not available" for extending the path.

2. Generating a directed circuit of q vertices q times—once at each vertex in the circuit—must be avoided. This is accomplished by insisting that no vertex $i \leq p_1$ is available for path extension, if the path begins with vertex p_1. This rule assures that the search for a particular directed circuit commences only when its lowest-numbered vertex is at the path initiation.

3. The same path must not be considered more than once during the path extension. This is accomplished by keeping an updated list of forbidden vertices in a binary n by n matrix $\mathsf{H} = [h_{ij}]$. The 1 entries in the ith row correspond to the vertices that are forbidden from vertex i(i.e., if vertex j is forbidden from vertex i, set $h_{ij} \leftarrow 1$). A 0 entry indicates that the vertex is not forbidden (i.e., if $h_{ij} = 0$, then vertex j is not forbidden

from vertex i). Matrix H is reset to zero each time a new vertex is chosen as the starting vertex.

The digraph is inputted as its adjacency matrix [see Section 11-2(a)]. The vertices are labeled as usual with integers 1, 2, . . . , n. The directed path under consideration is represented by a linear array

$$P = (p_1, p_2, \ldots, p_{k-1}, p_k, 0, 0, \ldots, 0, 0)$$

of order n. The first vertex of every path is p_1 and the last one is p_k.

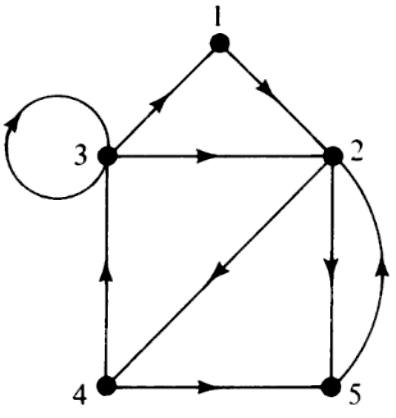

Fig. 11-6 Digraph.

The algorithm can be best explained with an example. When applied to the digraph of Fig. 11-6, the following steps will be performed:

Path P					
p_1	p_2	p_3	p_4	p_5	*Action on P*
1	0	0	0	0	
1	2	0	0	0	
1	2	4	0	0	
1	2	4	3	0	Report circuit: set $h_{4,3} \leftarrow 1$
1	2	4	0	0	
1	2	4	5	0	No circuit: set $h_{4,5} \leftarrow 1$
1	2	4	0	0	No circuit: $h_{4,1} \leftarrow h_{4,2} \leftarrow h_{4,3} \leftarrow h_{4,4} \leftarrow h_{4,5} \leftarrow 0$; $h_{2,4} \leftarrow 1$
1	2	0	0	0	
1	2	5	0	0	No circuit: $h_{2,5} \leftarrow 1$
1	2	0	0	0	No circuit: $h_{2,1} \leftarrow h_{2,2} \leftarrow h_{2,3} \leftarrow h_{2,4} \leftarrow h_{2,5} \leftarrow 0$; $h_{1,2} \leftarrow 1$
1	0	0	0	0	No circuit: $p_1 \leftarrow p_1 + 1$; H $\leftarrow 0$
2	0	0	0	0	
2	4	0	0	0	
2	4	3	0	0	Report circuit: $h_{4,3} \leftarrow 1$
2	4	0	0	0	
2	4	5	0	0	Report circuit: $h_{4,5} \leftarrow 1$
2	4	0	0	0	No circuit: $h_{4,1} \leftarrow h_{4,2} \leftarrow h_{4,3} \leftarrow h_{4,4} \leftarrow h_{4,5} \leftarrow 0$; $h_{2,4} \leftarrow 1$
2	0	0	0	0	
2	5	0	0	0	Report circuit: $h_{2,5} \leftarrow 1$
2	0	0	0	0	No circuit: $p_1 \leftarrow p_1 + 1$; H $\leftarrow 0$

Path P (Cont.)

p_1	p_2	p_3	p_4	p_5	*Action on P*
3	0	0	0	0	Report circuit: $p_1 \leftarrow p_1 + 1$; H $\leftarrow 0$
4	0	0	0	0	
4	5	0	0	0	No circuit: $h_{4,5} \leftarrow 1$
4	0	0	0	0	No circuit: $p_1 \leftarrow p_1 + 1$; H $\leftarrow 0$
5	0	0	0	0	No circuit: stop

The flow chart of this algorithm, which is a modified version of the algorithm given in [11-63], is shown in Fig. 11-7.

You must have observed that this algorithm is nothing more than a systematic and exhaustive search for directed circuits. As shown in the example, the same directed path is traversed many times. Even a directed circuit is usually examined and rejected several times before its turn to be accepted arrives. Consequently, the algorithm is very slow, and there is room for considerable improvement. To quote Tiernan [11-63], this algorithm "would be costly to utilize on a graph containing more than 50 arcs or 7 vertices"—a small graph indeed.

The algorithm could be easily modified to generate all directed Hamiltonian circuits. This, in fact, was the original purpose of the algorithm as reported by Roberts and Flores [11-56].

A random directed graph of 20 vertices, 55 edges, and 434 directed circuits took about 17 seconds on the IBM 7044. This indicates that this method, involving a systematic but exhaustive search, is quite inefficient in terms of execution time.

A similar algorithm but somewhat more involved, considerably faster, but requiring more storage was proposed by Weinblatt [11-67]. See also [11-50].

11-5. SHORTEST-PATH ALGORITHMS

A large number of optimization problems are mathematically equivalent to finding shortest paths in a graph. Consequently, shortest-path algorithms have been worked over more thoroughly than any other algorithm in graph theory. More than 100 papers have been published and dozens of algorithms have been proposed. Some of these algorithms are better than others, some are more suited for a particular structure than others, and some are only minor variations of earlier algorithms. For a good comparative study of various shortest-path algorithms through the year 1968, a survey paper by Dreyfus [11-17] is highly recommended.