D STATES
MENT OF
MERCE
CATION

NBS TECHNICAL NOTE 772

A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees

U.S.
RTMENT
OF
MMERCE
National
B eau
of
 rds

NATIONAL BUREAU OF STANDARDS

The National Bureau of Standards[1] was established by an act of Congress March 3, 1901. The Bureau's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the Nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau consists of the Institute for Basic Standards, the Institute for Materials Research, the Institute for Applied Technology, the Center for Computer Sciences and Technology, and the Office for Information Programs.

THE INSTITUTE FOR BASIC STANDARDS provides the central basis within the United States of a complete and consistent system of physical measurement; coordinates that system with measurement systems of other nations; and furnishes essential services leading to accurate and uniform physical measurements throughout the Nation's scientific community, industry, and commerce. The Institute consists of a Center for Radiation Research, an Office of Measurement Services and the following divisions:

Applied Mathematics — Electricity — Mechanics — Heat — Optical Physics — Linac Radiation [2] — Nuclear Radiation [2] — Applied Radiation [2] — Quantum Electronics [3] — Electromagnetics [3] — Time and Frequency [3] — Laboratory Astrophysics [3] — Cryogenics [3].

THE INSTITUTE FOR MATERIALS RESEARCH conducts materials research leading to improved methods of measurement, standards, and data on the properties of well-characterized materials needed by industry, commerce, educational institutions, and Government; provides advisory and research services to other Government agencies; and develops, produces, and distributes standard reference materials. The Institute consists of the Office of Standard Reference Materials and the following divisions:

Analytical Chemistry—Polymers—Metallurgy—Inorganic Materials—Reactor Radiation—Physical Chemistry.

THE INSTITUTE FOR APPLIED TECHNOLOGY provides technical services to promote the use of available technology and to facilitate technological innovation in industry and Government; cooperates with public and private organizations leading to the development of technological standards (including mandatory safety standards), codes and methods of test; and provides technical advice and services to Government agencies upon request. The Institute also monitors NBS engineering standards activities and provides liaison between NBS and national and international engineering standards bodies. The Institute consists of a Center for Building Technology and the following divisions and offices:

Engineering and Product Standards—Weights and Measures—Invention and Innovation—Product Evaluation Technology—Electronic Technology—Technical Analysis—Measurement Engineering—Building Standards and Code Services[4]—Housing Technology[4]—Federal Building Technology[4]—Structures, Materials and Life Safety[4]—Building Environment[4]—Technical Evaluation and Application[4]—Fire Technology.

THE INSTITUTE FOR COMPUTER SCIENCES AND TECHNOLOGY conducts research and provides technical services designed to aid Government agencies in improving cost effectiveness in the conduct of their programs through the selection, acquisition, and effective utilization of automatic data processing equipment; and serves as the principal focus within the executive branch for the development of Federal standards for automatic data processing equipment, techniques, and computer languages. The Center consists of the following offices and divisions:

Information Processing Standards—Computer Information—Computer Services—Systems Development—Information Processing Technology.

THE OFFICE FOR INFORMATION PROGRAMS promotes optimum dissemination and accessibility of scientific information generated within NBS and other agencies of the Federal Government; promotes the development of the National Standard Reference Data System and a system of information analysis centers dealing with the broader aspects of the National Measurement System; provides appropriate services to ensure that the NBS staff has optimum accessibility to the scientific information of the world, and directs the public information activities of the Bureau. The Office consists of the following organizational units:

Office of Standard Reference Data—Office of Technical Information and Publications—Library—Office of International Relations.

[1] Headquarters and Laboratories at Gaithersburg, Maryland, unless otherwise noted; mailing address Washington, D.C. 20234.

[2] Part of the Center for Radiation Research.

[3] Located at Boulder, Colorado 80302.

[4] Part of the Center for Building Technology.

A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees

Judith Gilsinn

Applied Mathematics Division
Institute for Basic Standards
U.S. National Bureau of Standards
Washington, D.C. 20234

and

Christoph Witzgall

Boeing Scientific Laboratories
P.O. Box 3999
Seattle, Washington 98124

NBS Technical Notes no. 772 are designed to supplement the Bureau's regular publications program. They provide a means for making available scientific data that are of transient or limited interest. Technical Notes may be listed or referred to in the open literature.

U.S. DEPARTMENT OF COMMERCE, Frederick B. Dent, *Secretary*
NATIONAL BUREAU OF STANDARDS, Richard W. Roberts, *Director*

Issued May 1973

National Bureau of Standards Technical Note 772

Nat. Bur. Stand. (U.S.), Tech. Note 772, 92 pages (May 1973)
CODEN: NBTNAE

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402
(Order by SD Catalog No. C13.46:772).
Price $1.25 domestic postpaid or $1.00 G.P.O. Bookstore.

TABLE OF CONTENTS

LIST OF TABLES

A PERFORMANCE COMPARISON OF LABELING ALGORITHMS FOR CALCULATING SHORTEST PATH TREES

Judith Gilsinn and Christoph Witzgall

Many applications in transportation and communication require the calculation of shortest routes between points in a network, and several algorithms for the solution of this problem exist in the literature. This paper examines one class of such algorithms, that which calculates a shortest route from one point in the network to all other intersection points. Computer data handling techniques which can be used to improve the two basic algorithms in this class are investigated. Results of computer timing runs on various types and sizes of networks are compared, and the differences, sometimes of an order of magnitude, are analyzed. Detailed flowcharts and computer programs of the tested algorithms are also included.

Key words: Algorithms; networks; paths; shortest-paths; trees.

1. Introduction

In many an analysis of communication and transportation systems, the problem of finding the shortest, cheapest, or fastest route between two points arises naturally. One may wish to determine, for instance, the least cost route to travel by auto between two cities or alternatively the auto route which is fastest. One may want to route a telephone call along telephone lines to minimize the length of the lines over which the call travels.

In other applications, the actual routing is less important than the numerical value of the minimum time, cost, or distance required. For instance, one may need to decide which of several warehouses is closest to his store by rail, in which case he is interested in the rail distances between his store and the various warehouses rather than the actual routing. One may wish to compare minimum travel times from suburban areas to the city center by various transportation modes, or to find the minimum number of relay points which a message has to go through as it is sent from one point in a communication network to another.

In all of the examples given above one has a network, which may consist of road segments, railroad track, other common carrier transportation routes, or direct communication links, and one desires a routing between two points in this network which is minimum according to some criterion. This problem is usually termed that of finding a shortest path in the network. (The terms "path" and "network" will be defined more precisely in the next section, but their common understanding is sufficient for the present discussion.)

Often in applications requiring the finding of shortest paths a great many such paths are desired. Typically one wishes to find shortest paths from all of some subset of network intersections to all of some other (possibly the same) subset of intersections, where "intersections" can be switching or relay points in a communication network, road intersections, railroad junction points, airplane terminals, etc. For many applications the networks are very large, containing many segments and intersection points, thus requiring the use of a digital computer for their processing.

A number of algorithms have been developed, and appear in the literature, for finding shortest paths in such large networks. It is natural to compare the computational efficiency of the different algorithms on different types of networks. Dreyfus [12]* has written an excellent survey paper classifying the types of algorithms and giving theoretical computational bounds for each class. Martin [35] and Hitchner [21] have conducted computational experiments on several algorithms.

These previous studies, however, have only sketched the actual computer implementation of the algorithms included, whereas it has been our experience,confirmed by the results reported in Table 1 of section 7,

* Figures in brackets indicate the literature references on page 71.

that the data handling (more specifically, list processing) techniques employed in programming the algorithms for a digital computer have a great effect on computation time. Therefore, it is the purpose of this paper to examine one class of shortest path algorithms, to provide a unified structure for describing and coding them, to give list processing techniques which can be used to improve the efficiency of the basic algorithms, and to report the results of computational experiments utilizing these techniques in such a manner that the exact computer implementation of the algorithms is known.

This report is directed primarily to the reader who is confronted with a problem requiring shortest path calculation and who does not have the time or resources to locate or evaluate available algorithms. Some familiarity with computers is assumed, but the paper is intended to be self-contained in terms of network definitions and algorithm descriptions.

Sections 2 and 3 below give the basic definitions and background relating to the network structures appearing in the algorithms. Section 4 delineates the class of algorithms to be examined in the remainder of the report and describes the two basic algorithms in this class. Sections 5 and 6 give list processing techniques which can be used to improve the efficiency of the basic algorithms*. Section 7 includes the results of the computational experiments and a discussion of the relative merits of the various techniques. Section 8 describes conventions used in the flowcharts of the algorithm implementations which are included in section 9 together with detailed descriptions of the implementations. The actual programs appear as an appendix.

* Other techniques (not reported here but described in an earlier, unpublished work by Witzgall) have also been tested, and analyses of these tests coupled with hindsight have revealed that the techniques reported in this paper are the most likely candidates for success.

2. NETWORK REPRESENTATION

In order to give precise descriptions of the shortest path algorithms, it is first necessary to formalize some definitions. A network consists of a finite set V of nodes and a finite set E of arcs. To each arc $e \varepsilon E$ there corresponds an ordered pair (v,w) of nodes, the starting or beginning node v and the terminating or ending node w, and we say the arc e leads from v to w. The networks thus defined are directed networks, since the distinction between the beginning and ending of an arc imposes a direction on each arc. The term "network" without modifier will mean directed network throughout this text.

A directed path, or simply path, is a sequence

$$P = (e_1,e_2,\ldots,e_n)$$

of arcs such that for each $i = 2,\ldots n$, arc e_i begins at the end of arc e_{i-1}. Let v be the beginning of arc e_1 and w the end of arc e_n; then we say the path P starts at v and ends at w, and that P is a path from v to w. If a network contains a path from node v to node w, then w is called accessible from v. If $v = w$, i.e. path P starts and ends at the same node, P is called a circuit. A path for which $e_i \neq e_j$ whenever $i \neq j$, i.e. no arc appears in P more than once, is called arc-simple.

Assume now that each arc e of a network N has a non-negative length $\ell(e)$ associated with it. Then each path P can be assigned a path length

$$d(P) = \sum_{i=1}^{n} \ell(e_i) ,$$

and we may speak of a shortest path from one particular node to another as a path P for which $d(P)$ is minimum. Note there may be several such paths, all having the same (minimum) length; the algorithms described below only find one of them.

An alternative representation of path P would be the node sequence

$$P' = (v_1,v_2,\ldots,v_{n+1})$$

where for each i, arc e_i starts at node v_i and ends at node v_{i+1}. P' is a unique representation of P if and only if each arc e is uniquely determined by its starting and ending nodes, i.e. if for each node pair (v,w) there is at most one arc from v to w. In this paper we are interested only in shortest paths. Therefore we can assume without loss of generality that at most one arc leads from v to w, since if multiple such arcs exist only a single minimum length arc among them is needed for shortest paths between any node pair.*

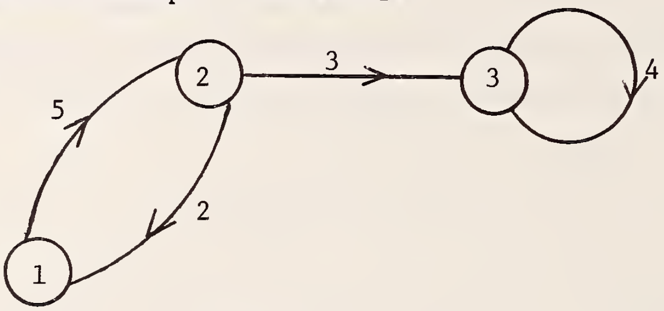
A network containing M arcs and N nodes may be represented in the computer in several ways. One way, which we will call the matrix representation, considers all possible pairs of nodes (v,w) and stores for each pair either the length of the arc from v to w, or ∞ if no such arc exists. This information may be stored in an N by N matrix. However, for many applications N is quite large and M is small compared to N^2, so that most of the entries in the matrix are ∞. In this case storing the entire matrix in the computer is inefficient, and if N is large it may well be impossible.

A second way of representing a network, which will be called the ladder representation, requires listing all the arcs in the network. For each arc we need to know its starting node, its ending node, and its length. Therefore the ladder network representation requires $3M$ storage locations. However, if the arcs in the network are ordered by starting node so that all the arcs starting at the same node appear together, only $N + 2M$ storage locations are needed, since one only needs to know which is the first arc starting at each node v. One can then determine the last arc starting from v as the arc immediately

* The programs for the shortest path algorithms described in this paper, however, do calculate correct path lengths when multiple arcs exist for the same node pair.

preceding the first arc starting at node v + 1. (This of course assumes that the nodes are numbered sequentially from 1 to N . If the nodes do not naturally appear in such a fashion, it will be necessary to convert them to this form since efficient operation of the algorithms requires that nodes be numbered sequentially.) The representation of a network in which all the arcs starting at the same node appear together is called forward star form, and the forward star of node v consists of all arcs starting at v .

Consider for example the network



where the node numbers appear in circles and the arc lengths are presented next to the arcs to which they apply. This network may be represented in ladder form by the following lists.

arc number	beginning node	ending node	arc length
1	1	2	5
2	2	1	2
3	2	3	3
4	3	3	4

(Note that only the last three columns require computer storage locations.)

The same network is represented in forward star form by the following lists.

arc number	ending node	arc length
1	2	5
2	1	2
3	3	3
4	3	4

node number	first arc starting at this node
1	1
2	2
3	4

(Again the columns headed "arc number" and "node number" do not actually require computer storage locations.) Two of the lists, that labeled "ending node" and that labeled "arc length", are the same in the two forms. The third list (labeled "beginning node") required by the ladder representation is replaced in the forward star form by the list giving the number of the first arc starting at each node.

Corresponding to the two fundamental ways of storing a network are two basic methods for treating shortest path problems: *matrix* methods which use the matrix representation of the network and *labeling* methods which utilize ladder or forward star representations.

Matrix methods yield the shortest distances between all pairs of nodes simultaneously. Because of their large storage requirements their application is restricted to relatively small networks. The computational effort of matrix methods depends only on the number of nodes and is independent of the actual number of arcs present in the network. For this reason, matrix methods tend to be less efficient if only a relatively small percentage of possible arcs in a network are realized. Most large practical networks are, however, of this kind. It is for this reason, that this paper concentrates on labeling methods.

3. TREES

The purpose of this paper is to categorize labeling algorithms for determining shortest paths, to identify some of the "strategies" characterizing these algorithms, to formalize their descriptions, and to document computational experiments aimed at assessing the relative merits of the various algorithms for different types of networks. This section will contain some additional definitions required for describing network structures as used by the labeling algorithms. The basic labeling algorithms themselves will be given in the next section.

In the context of directed networks, a rooted tree, or simply tree, is a network T together with a node r (the root), such that each node of T, except r, is accessible from r by a unique arc-simple path in T. Alternatively the network T is a tree if

1.) every node of $T-\{r\}$ is the end of exactly one arc in T,

2.) r is not the end of any arc in T, and

3.) there are no circuits in T.

We call a rooted tree T a minimum tree (in the larger network under discussion) if T contains all nodes accessible from r, and if for each node v in T, the unique path in T from r to v is a shortest path from r to v in the network. The phrase shortest path tree has the same meaning.

The standard shortest path problem for a network reads:

Problem 1 Find a shortest path from a given node r to a given node v. A second, related problem reads:

Problem 2 Find a minimum tree rooted at r. Such a tree always exists, although it may consist of the node r alone.

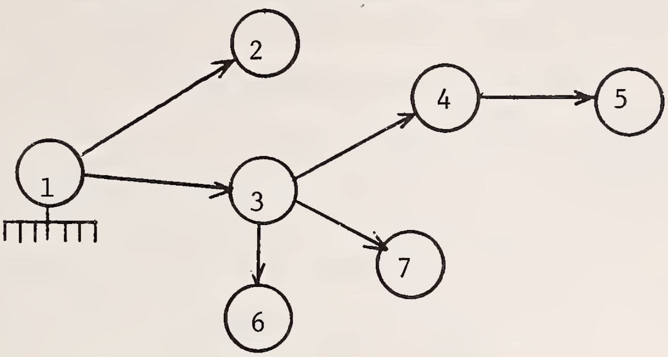
The labeling algorithms described in this report solve Problem 1 by actually solving Problem 2, thereby simultaneously determining the shortest paths from r to all nodes accessible from r.

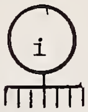
Conditions 1 and 2 of the second definition of "rooted tree" given above insure that the tree T can be described in a computer by a list of length n-1, where n is the number of nodes, including r, in the tree.* This list contains, for each node $w \neq r$ in the tree, the starting

* If all network nodes are accessible from r, then $n=N$ the number of nodes in the network.

node v of the single arc in the tree terminating at w .

Consider the following tree.



(Note the symbol  indicates that node i is the root. This tree can be described by the following "predecessor list".

node number	predecessor
1	0*
2	1
3	1
4	3
5	4
6	3
7	3

In the algorithms which will be described below, another list, indexed by the node numbers and associated with the tree T , is required. This list contains for each node v a label d(v), the length of the single path from r to v in T . Nodes not in T may or may not be labeled; usually they are given the label ∞ . The root r has label 0.

* We have used the zero here to denote the fact that node 1 is the root of the tree.

4. BASIC LABELING METHODS

Labeling methods for computing shortest paths can be divided into two general classes, label-correcting and label-setting methods.

The typical label-correcting method starts with any tree T rooted at r and "corrects" T until no further improvement or enlargement is possible. Label-setting methods start out with the tree T consisting of r alone, and augment T one arc at a time in such a manner that at each step T is a minimum tree for all nodes in T. Termination for a label-setting method occurs when all arcs which start in T also end in T, or --- if the goal was to determine the shortest paths only to a subset S of nodes --- when all nodes in S are in the tree.

We now give a more precise description of each of these two methods. In both methods each stage of the algorithm is characterized by a tree T rooted at the node r. In the descriptions below $\ell(e)$ is the length of arc e and $d(v)$ is the label of node v, equal to the length of the unique path in T from r to v.

The basic label-correcting method is as follows:

(1) Start with any tree T rooted at r and with labels $d(v)$ equal to the length of the path in T from r to v whenever such a path exists; otherwise $d(v): = \infty$ *. If no other tree is readily available, set T to be the tree consisting of the root node r alone (and no arcs), and set

$$d(v): = \begin{cases} +\infty \text{ if } & v \neq r \\ 0 \text{ if } & v = r . \end{cases}$$

* The notation $a: = b$ will be used to denote the operation of transferring the contents of computer memory location b to the location a.

(2) Find an arc e from node s to node t, such

$$d(s) + \ell(e) < d(t) .$$

Then redefine

$$d(t): = d(s) + \ell(e) ,$$

and adjoin e to T, removing any other arc in T which ends at t.

(3) Repeat (2) until, for all arcs e

$$d(s) + \ell(e) \geq d(t) .$$

The tree T will then contain a shortest path from r to all nodes accessible from r, and for each node v, $d(v)$ will be the length of a shortest path from r to v.

The basic <u>label-setting method</u> is as follows:

(1) Set $T: = \{r\}$ and $d(r): = 0$.

(2) Among all arcs e which begin at a node s in T and end at a node t not in T, find one that minimizes $d(s) + \ell(e)$. Put that arc into T and define

$$d(t): = d(s) + \ell(e) .$$

(3) Repeat (2) until all arcs that start in T also end in T.

The tree T and the labels $d(v)$ then contain the shortest paths and their lengths as above.

It should be noted that label-setting methods work only for non-negative arc lengths. If there are no circuits of negative length present, then label correcting methods will still work even if there are some arcs of negative length.

Flowcharts for the computer implementation of these two basic algorithms appear later in section 9, in addition to variations on these two basic methods which utilize the list processing techniques described in the next two sections.

5. TECHNIQUES TO IMPROVE THE LABEL-CORRECTING METHOD

In the description of the basic label-correcting method, step (2) involves finding <u>any</u> arc e which can be added to the tree with a decrease in the label of its terminal node t . The first observation which can be made to improve this basic algorithm is that one needs to look for such an arc only among those arcs whose starting node s is already labeled with a finite number. The implementation of the basic label-correcting algorithm described in section 9 (as algorithm C) uses this observation. In fact, none of the algorithm implementations given in this paper actually examine the whole arc list at each step; all focus on arcs in the forward star of a node which meets certain criteria which depend on the algorithm. The order in which forward stars of nodes are examined is a major factor in the efficiency of the algorithm, and the techniques reported below are mainly aimed at obtaining an efficient ordering.

One observation which can aid in determining which forward stars should be examined (and when) during the course of the algorithm, is that one need examine a node only if it has not been examined since its label was last changed. This can be accomplished through the use of an <u>alteration flag</u> for each node, which is set when the label to that node is changed and remains set until the entire forward star of that node has been examined. Algorithm C1 , described in section 9 below, implements this process. In each pass of algorithm C1 , the node list is searched in order and the forward star of each node whose flag is set is examined for arcs which should be included in the tree. Termination occurs when no more flags are set.

The forward stars of nodes need not be examined in numerical order as above; they may instead be examined in the order in which the nodes were labeled. That is if node v_1 was labeled before node v_2, then the forward star of v_1 is examined before that of v_2, regardless of the node numbers v_1 and v_2. The technique employed here is the sequence list. Nodes are placed on the sequence list as their labels are altered, and removed from the list as their forward stars are examined. If the forward stars of nodes are examined in the order in which they are placed on the sequence list, the list is said to be treated in a FIFO (First-In, First-Out) manner; if the forward star of the latest node added to the list is examined before that of a node placed on the list previously, it is said to be treated in a LIFO (Last-In, First-Out) manner. In general examining the list in a FIFO manner is much better than the LIFO alternative, since nodes in some sense "closest" to the root are examined before those further out in the tree.* No alteration flag is necessary when using a sequence list since one examines the forward stars only of those nodes which appear on the list. In algorithms utilizing such a list, termination occurs when the list is empty.

There is one problem which may arise in using a sequence list. If a node is placed on the list whenever its label is changed, the same node may appear in more than one position on the list. This means that the number of list positions required may be more than the number of nodes. One way to avoid this situation is to use a flag indicating, for each node, whether or not it is already on the list. A node already on the list is not added a second time. If nodes appear at most once on the list, the length of the list may actually be considerably less than the number of nodes, since positions are required only for the maximum number

* If a path in T is extended from its end node before the labels of nodes closer to the root have been lowered, the extension will have to be relabeled later on.

of nodes on the list at any one time. The list may be treated in a "rotation" manner through the use of two pointers u and v, where u points to the entry whose forward star is to be examined next and v is the position of the last node added. As u moves down the list, there is unused space at the top of the list above u which may be used for new nodes when space at the bottom of the list is exhausted. Algorithm C2, described in section 9 below, employs a FIFO sequence list on which nodes appear at most once and which is treated in this rotating manner.

By the branch $B(v)$ attached to v, we mean the set of all nodes $w \neq v$ in T for which the unique path in T from r to w contains v. Another observation which can be used to improve label correcting methods is that if the label of node v is decreased by an amount Δ the labels of all nodes in the branch attached to v may be decreased by Δ. It is still necessary to reexamine the forward stars of these nodes since their labels have been changed, but the reexamination is done using the new lower label. Algorithm C3 utilizes this observation by employing a forward pointer f which arranges the tree nodes in order so that nodes on the same branch appear together. Pointer f has the following properties:

1) $f(v) = r$ for exactly one node v,

2) for every node in the tree, there is an integer k such that

$$f^k(r) = v \text{ , and}$$

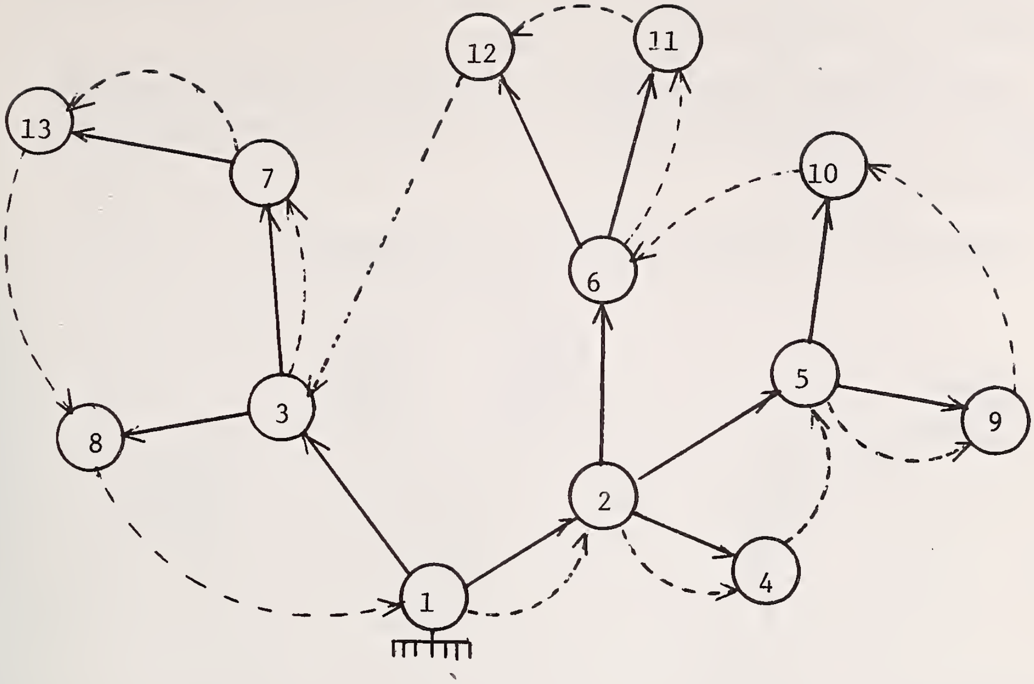
3) for every branch $B(v)$

   a. for each node $s \varepsilon B(v)$, there is an integer j such that

   $$f^j(v) = s \text{ , and}$$

   b. if $B(v) \neq \Phi$, then there is a unique node $w \varepsilon B(v)$ such that $f(w) \notin B(v)$.

The dotted lines in the following diagram illustrate such a forward pointer f.



All such forward pointers have the property that if v is on the path in T from r to w then there is an integer k such that $f^k(v) = w$. This follows from property 3a and the definition of a branch.

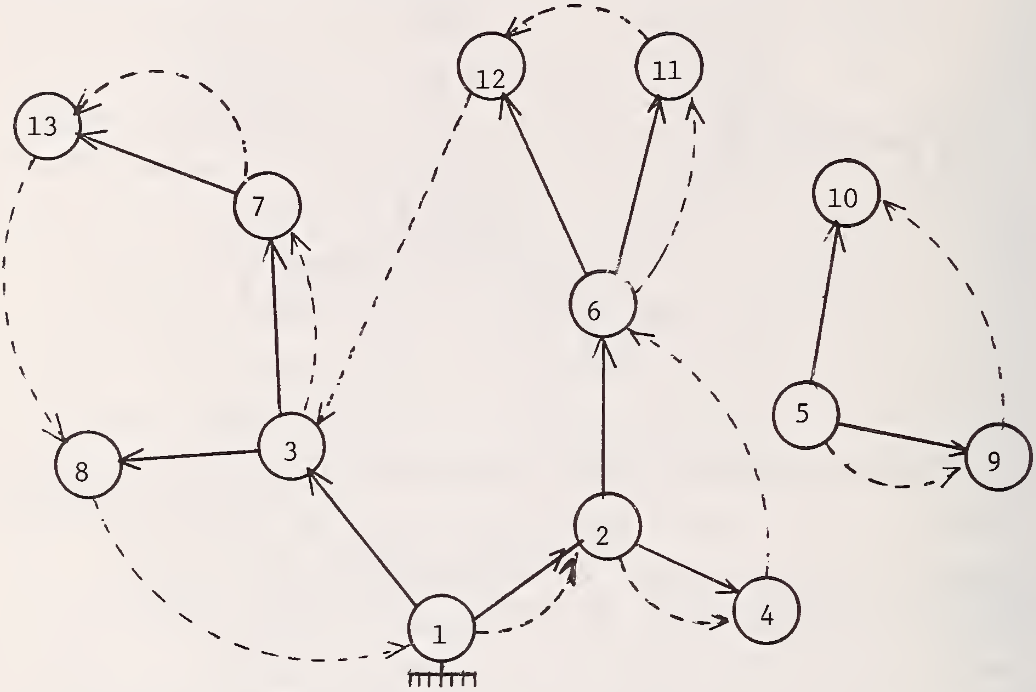
When the label to a node w is changed, in addition to decreasing all labels on the branch attached to w, it is necessary to detach the branch together with w from the old predecessor v_1 of w and to reattach them to the new predecessor v_2 of w. This necessitates a change in the forward pointer.

Suppose w and the branch $B(w)$ are to be detached from v_1. If $w = r$, the empty tree will result. If $w \neq r$, there is a node s such that $w = f(s)$. One finds s by starting at the old predecessor v_1 of w. Since w is in $B(v_1)$, there is some k such that $f^k(v_1) = w$, and the node $s = f^{k-1}(v_1)$ is such that $f(s) = w$. Let t be the "last" node in the branch $B(w)$; that is, t is in $B(w)$ but $f(t)$ is not. (Property 3b insures that t exists.) Then w and the branch $B(w)$ may be detached from the tree by setting $f(s) := f(t)$, and removing the pointer f from t.

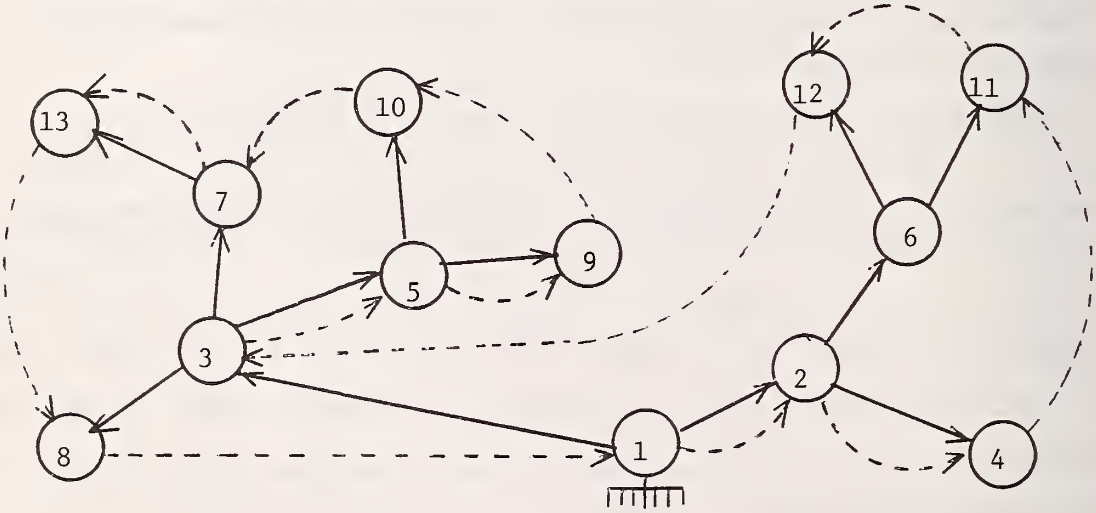
Attaching the branch B(w) together with the node w to the node v_2 involves setting $f(t) := f(v_2)$ and $f(v_2) := w$, as well as updating the predecessor of w to be v_2 .

As an example, node 5 and its branch {9,10} in the preceding diagram are detached from node 2 and reattached to node 3 in the following two steps:

1. Detach node 5 and its branch {9,10} from node 2.



2. Reattach node 5 and its branch {9,10} to node 3.



The basic label-correcting method has been ascribed to Moore [41], and appears both in Ford and Fulkerson [15] and Berge [4]. Bellman [1] has described a slight variant. The alteration flag technique used in C1 is described by Bock, Kantner, and Haynes [5] and also by Hoffman and Pavley [23].

6. TECHNIQUES TO IMPROVE THE LABEL-SETTING METHOD

Sequencing techniques and lists are also used to improve the basic label-setting algorithm. In the process of finding an arc from a node in the tree to one outside the tree which yields the "minimum tree extension" (Step 2 of the basic label setting method; see section 4), many possible labels are calculated and discarded. The improvements described below all involve retaining this information (as temporary labels) to avoid recalculation. Indeed, the algorithms are designed so that each arc is examined only once.

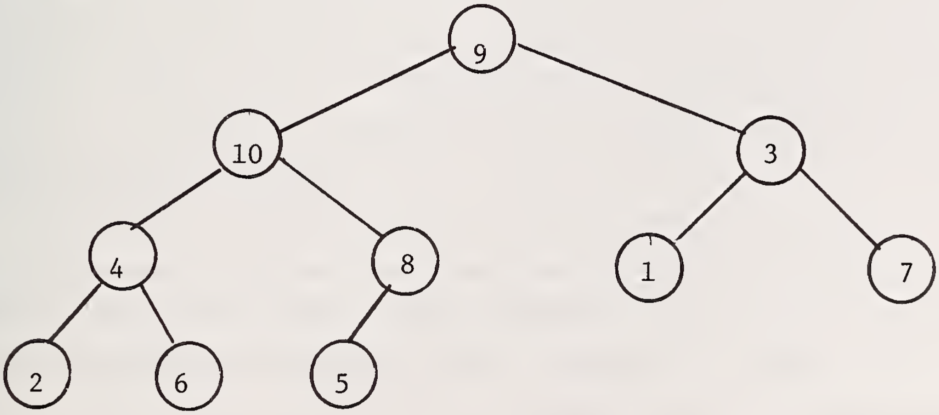
The first improvement, implemented in algorithm S1, involves keeping a sequence list of nodes not yet in the tree linearly ordered by their temporary labels. Each of the nodes in the sequence list is thus accessible from at least one node in the current tree by a single arc, and its position in the list is determined by the label it would have were that arc to be adjoined to the tree. If there is more than one such label for one node in the list, its position is determined by the minimum label. Thus if the temporary label for a node v is lowered because of an addition to the tree, v must be moved up in the sequence list. Adding a node to the list or moving one to a new position involves searching the list to find the position which that node should occupy. At each step the top node on the list, i.e. the node which has smallest temporary label, receives its permanent label (equal to its latest temporary label), and its forward star is the next to be examined.

This procedure consumes a great deal of time in finding the position a node should occupy when it is first added to the list or when its temporary label is changed. The other algorithms reported here are designed to reduce the time spent in these actions. In the first modification, implemented in algorithm S2, the list is partially ordered in a <u>tree sorted form</u>, rather than linearly ordered. The elements of such a list may be considered nodes in a <u>binary tree</u> (not to be confused with the minimum tree). A binary tree containing n nodes is a tree in which for some $k \leq n$, each node numbered less than k has exactly two arcs starting at it, node k has either one arc or no arcs starting at it, and nodes k+1 through n have no arcs starting at them. The elements of a tree sorted list can be considered nodes of a binary tree in which the label of the predecessor of any node never exceeds the label of the node. The tree nodes can be sequentially written in successive horizontal layers, the first being of cardinality 1, the next of cardinality 2, the next of cardinality 4, and the k^{th} of cardinality 2^{k-1}. The labels of nodes in any path in the binary tree are linearly ordered, but nodes on different paths are non-commensurable.

Suppose for example that one has the following nodes and labels in the shortest path tree, and wishes to keep the nodes in a list tree-sorted on label:

node	label
1	5
2	4
3	4
4	2
5	9
6	10
7	12
8	3
9	1
10	2

One may construct the following binary tree having the property that the label of the predecessor of any node in the binary tree never exceeds the label of a successor in the tree.



This binary tree is stored in list form in the computer as follows on the left.

tree list	labels
9	1
10	2
3	4
4	2
8	3
1	5
7	12
2	4
6	10
5	9

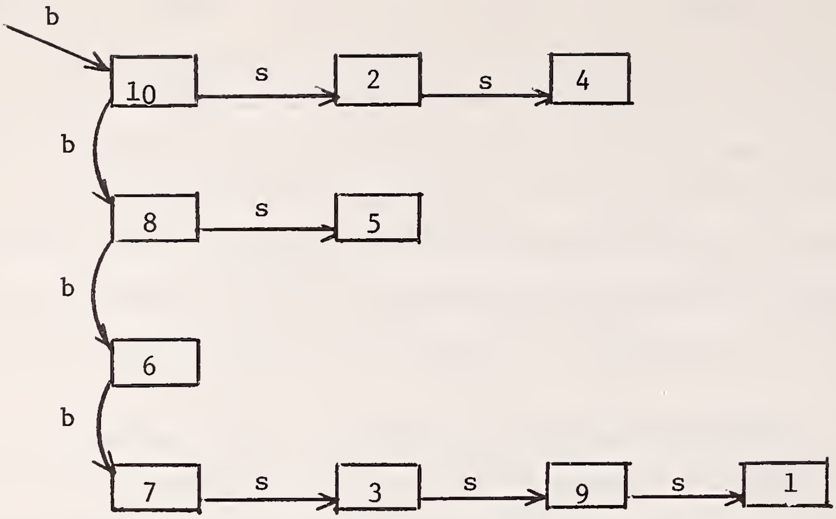
Although the list is in tree sorted form, a glance at the associated list of labels on the right shows that it is clearly not in linear sorted form.

Tree ordering guarantees that the top element in the list has minimum label, but tree ordering requires much less work to establish and update then linear ordering. In fact, if there are n elements in the list, the tree sorting procedure only requires at most k comparisons, where k is the first integer such that $2^{k+1} > n$, to find the position to be occupied by a new list entry. To accomplish this, the new entry is tentatively put in the (n+1) st position. It is then compared with the entry in the position (n+1)/2 (which is in the next higher horizontal layer), and is interchanged with that entry if it is less. This interchange process is continued until the new entry is greater than or equal to some old entry and no interchange takes place, or else the new entry becomes the top entry in the list. Although adding a node to the tree-sorted list requires less work than adding to a linearly sorted list, the tree-sorted list requires additional reordering when the top element is removed. This procedure, too, takes no more than k steps where k is as above.

The implementations of the algorithms described in section 9, require that arc lengths be non-negative integers. Label-correcting algorithms will work correctly if arc lengths are negative, so long as no negative-length circuits exist, but label-setting algorithms will not work correctly on such networks. The integrality requirement is necessary only for algorithm S4, but in practice this requirement is not unduly restrictive for the other algorithms as well. In most practical applications using shortest path algorithms, the arc lengths are either integers or can be transformed to integral values by using an appropriate scale factor.

Once this transformation has taken place, if the maximum arc length is small (say compared to the number of nodes), advantage can be taken of the fact that many of the temporary labels are the same. This equality of labels arises from the further observation that the difference between the largest and smallest temporary labels is at most the length of the longest arc (this is true because, for label-setting algorithms, the smallest temporary label is at least as large as any permanent label, and a temporary label is the sum of a permanent label and one arc length). Therefore if the number of temporary labels on the list is large compared with the maximum arc length, one would expect many such labels to be equal.

Algorithm S3 of section 9 takes advantage of this fact by "chaining ties" in such a manner that when the sequence list is searched for the position to be occupied by a new entry, the label of only one of the nodes in a chained group need be examined since all others have the same label. The chaining process is accomplished by having a pointer (b in the example below) to the first element of the first chain and the first element of each succeeding chain and another pointer (s in the example below) to successors of elements within a chain. These are illustrated in the following example.



The numbers in the boxes represent node numbers. To find the position to be occupied by a new node, one need only check through the sequence of nodes given by the pointer b . In the example, this means checking at most four entries rather than possibly ten.

Algorithm S4 of section 9 takes more direct advantage of the observations above. Instead of using a sequence list in which the position of a new entry is determined by a search or sorting procedure, this algorithm puts a node in a list position determined directly by the temporary label of the node, modulo the maximum arc length plus 1. Since several nodes may have the same temporary label, this list also uses the chaining procedure described above. In addition it is treated in a rotating fashion, using a single pointer to the position of the current top node, to eliminate having to move chains up the list whenever the current maximum permanent label is changed.

The basic label-setting algorithm is credited to Minty by Pollack and Wiebenson [39], and is also described by Hu [26]. A slight variant is described by Dijkstra [11] and by Whiting and Hillier [61]. The tree-sorting technique used in algorithm S2 appears in Knuth [30]. Algorithm S4 is credited to Loubal by Hitchner [21] and also appears credited to Dial [10].

7. EXPERIMENTAL RESULTS

The seven algorithms C1, C2, C3, S1, S2, S3, and S4 were programmed from the flowcharts given in section 9 in FORTRAN V (UNIVAC's expanded FORTRAN IV), and were run under the EXEC II operating system on the UNIVAC 1108 at the National Bureau of Standards. This computer has an effective cycle time of 375 nanoseconds and 65,536 36-bit words of core storage of which about 53,000 words are available to the user. Copies of the programs appear as an appendix. The FORTRAN language was chosen because its wide availability makes it probable that in many applications of shortest path algorithms, this computer language would be used. No attempt has been made to take full advantage of peculiarities of the UNIVAC FORTRAN V compiler in order to obtain the fastest version of any program. However, it is hoped that because all programming was done by one person and the algorithms all used the same network structure, none of the algorithms was given a special advantage over the others resulting from any programming quirks.

In the experiments described below, all seven algorithms were run to compute the same trees in the same networks. The UNIVAC 1108 internal clock was interrogated before and after computation of each tree to obtain the actual computation time for that tree. No input or output time is included in these timings, but time for the algorithm initialization steps is included since it is part of the time required for computation of each tree.

Original timings (not reported here) of the shortest path algorithms were performed on highway networks prepared for use by the U.S. Department of Transportation's Northeast Corrider Transportation Project. It soon became clear that these medium-sized networks were too small to provide meaningful time distinctions on the UNIVAC 1108 computer. In addition, these networks all had arc/node ratios (the average number of arcs starting at a node) of between 3 and 4, which is typical of highway networks, whereas it was our purpose to compare the performance of the algorithms on a wider range of types of networks. For these reasons it was necessary to devise programs to generate a variety of quite large network types. Three computer programs were written and have been used to produce the networks upon which the comparative timing runs were made. All three programs create directed networks with the nodes numbered sequential from one to the number of nodes, and with (integer) arc lengths randomly assigned between zero and a desired maximum arc length (MAX ℓ in Table 1).

The first program creates a p by q rectangular <u>grid</u> network whose nodes are numbered sequentially from left to right and then top to bottom. The number N of nodes is thus pq and the number M of arcs is $4pq - 2p - 2q$. Since arc lengths are randomly assigned, the length of arc (a,b) is unlikely to equal the length of arc (b,a). Thus, although the networks created by this program have the connectivity of a grid, the triangle property may not hold and distances are not symmetric.

A second program creates a <u>complete</u> network on N nodes, that is, a network containing an arc from each node to each other node, $N \cdot (N-1)$ arcs in all.

The final network-generating program creates <u>random</u> connected networks with a specified number of nodes N and arcs M. The program first creates a spanning tree rooted at node 1 (i.e. a tree in which all other nodes in the network are accessible from node 1). This is done by starting with a tree containing only the root, and augmenting it at every stage by an arc (a,b), where a is a node chosen at random from those nodes already in the tree and b is chosen at random from those nodes not in the tree. The final $M - N + 1$ arcs are then obtained by randomly choosing pairs of nodes from the set of all possible node pairs.* Starting with a spanning tree rooted at node 1 insures that all nodes are accessible from <u>that</u> node at least. In practice this has been true as well from other nodes chosen as root, and timings are included only for trees for which this was so.

* Although the discussion in section 2 excluded networks containing multiple arcs for the same node pair, the computer programs given in the appendix to this paper actually calculate correct shortest path lengths for such networks. The generation procedure for random connected networks therefore allows multiple arcs, since additional effort is required to exclude them.

Table 1 gives timings for the seven algorithms as applied to 13 different networks created by the three programs described above. (Storage considerations are discussed later.) Each timing number reported in the table is the average of the times for five separate trees (corresponding to five roots) in that network.* For all the grid networks the five root nodes chosen were 3, 366, 729, 1092, and 1455 to insure that the trees were computed from five quite different network positions. For all the complete and random networks trees were computed from the nodes numbered 1 through 5. It should be remembered that the absolute magnitude of the time to compute a tree will vary from computer to computer; still the relative times should be rather similar for different machines.

The 13 networks used in the tests reported in Table 1 are believed representative of the types of large networks found most often in the shortest-path applications for which labeling algorithms are most suitable. Those networks with arc/node ratios between 3 and 4 are typical of large ground transportation (auto or rail) networks, while those with ratios between 5 and 20 are representative of many communications and air transportation networks. Networks containing fewer than one or two hundred nodes can be handled using matrix algorithms, since the whole matrix of distances can be stored at one time in the computer. Real-world applications in which there are more than a few hundred nodes and the matrix of arc lengths is not sparse are seldom found primarily (one supposes) because of the difficulties in gathering, checking, and processing so many data.** For instance, the complete network on 500 nodes would contain about 250,000 arcs. If even 10 percent had to be included as separate pieces of input data, 25,000 data points would have to be coded and checked for accuracy.

* Additional computational experiments, not reported here, support the representativeness of the times in Table 1. Also the variation among the times for the computation of each of the five trees by the same algorithm is generally less than the differences between algorithms.

** In addition, economy considerations work against designing real-world networks with much multiple connectivity.

TABLE 1

RESULTS OF COMPARATIVE TIMING EXPERIMENTS

		NETWORK				TIMINGS (SECS/TREE, AVERAGED FOR 5 TREES)						
NUMBER	TYPE	DESCRIPTION	NODES	ARCS	MAX ℓ*	C1	C2	C3	S1	S2	S3	S4
1	GRID	50×50	2500	9800	100	1.33	1.17	2.03	2.66	1.41	1.77	0.53
2	GRID	25×100	2500	9750	100	2.28	1.95	2.37	1.75	1.29	1.37	0.54
3	GRID	20×125	2500	9710	100	4.57	2.95	2.93	1.44	1.23	1.22	0.55
4	GRID	10×250	2500	9480	100	5.26	4.69	4.50	0.97	1.07	0.93	0.58
5	GRID	5×500	2500	8990	100	9.24	8.25	8.13	0.66	0.87	0.70	0.67
6	RANDOM		500	10000	100	0.62	0.60	0.77	3.47	0.58	0.90	0.33
7	RANDOM		1000	10000	100	0.69	0.67	0.89	9.35	0.93	1.89	0.40
8	RANDOM		2000	10000	100	0.73	0.70	0.96	19.12	1.57	2.95	0.49
9	RANDOM		3000	10000	100	0.80	0.74	1.02	26.88	2.19	3.88	0.57
10	COMPLETE		100	9900	100	0.40	0.47	0.49	0.49	0.32	0.42	0.26
11	COMPLETE		100	9900	100	0.37	0.43	0.47	0.50	0.37	0.42	0.27
12	GRID	5×500	2500	8990	1	3.33	0.36	7.50	0.42	0.73	0.57	0.44
13	RANDOM		3000	10000	3000	0.82	0.72	1.04	29.02	2.23	18.98	0.77

* MAX ℓ is the maximum arc length.

Algorithm S4 is clearly the fastest --- often by a wide margin --- in most of the cases reported in Table 1. In addition, its time is most consistent, depending primarily on the number of nodes rather than on network configuration. The speed of algorithm S4 is undoubtedly due to being able to enter nodes directly on the distance list with no sorting at all. However, accuracy requirements or storage limitations may make it impossible to set up this distance list, and may thus make other candidates more suitable in some applications.

The label-correcting method depends, in some sense, on the number of arcs in the shortest path having the greatest number of arcs, since in general at the k^{th} pass one is attempting to extend paths with k-1 arcs. Therefore label-correcting algorithms require much longer time for computing trees in "long thin" networks, such as the 5 by 500 grid (network 5 in Table 1) in which the shortest path from one of the corner nodes to that diagonally opposite it contains at least 505 arcs, than for a more "round" or "square" network, such as the 50 by 50 grid network (network 1 in Table 1) in which a similar shortest path would contain only about 100 arcs.

The label setting algorithms, on the other hand, depend primarily on the number of nodes in the network, since at each pass a new node is added to the tree. Of course the extra sorting required by some of the algorithms greatly increases the number of steps in a single pass. This seems particularly true for random connected networks in which the arc/node ratio is small. In such networks there are not many alternate paths to the same node, so that much of the sorting required by the label-setting algorithms does not contribute to a decrease in the number of steps required to find a shortest path tree. As an example, if the original network were actually a tree rooted at r , then the order in which forward stars are examined does not matter so long as those for nodes in the tree are examined before those for nodes not yet in the tree. In this case all sorting is unnecessary.

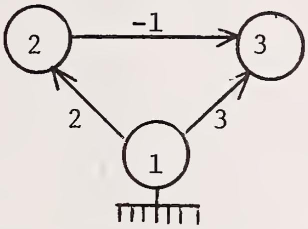
A shortest path in network 12 is essentially one which has fewest possible arcs in it, since all arc lengths are exactly 1. (Zero length arcs were allowed in the other networks, but not in this one.) Algorithm

C2 performed particularly well on this network, since in this case each node appears on the sequence list exactly once, and the list is ordered in such a manner that nodes at the top of the list are one arc from r , nodes directly below are 2 arcs from r , etc. In algorithm C1 nodes are examined in numerical order, rather than in the order in which they were labeled, necessitating more than one pass through the list. Algorithm C3 requires much time in setting up the lists for detaching and reattaching branches, a situation which never need occur for this network. In addition, sequencing by the inverse of the forward pointer means that in some cases nodes further out on a branch are examined before nodes closer to the root, an undesirable condition for this network.

Network 13 has the same connectivity properties as network 9 (i.e., if arc (a,b) exists in network 9, arc (a,b) also exists in network 13 and conversely), but it has different, and in general longer, arc lengths. As can be seen from Table 1, algorithm S4 begins to lose its advantage over label-correcting methods when the distance list,whose length is determined by the length of the longest arc, has as many entries as the number of nodes. The algorithms S1, S2, and S3 which require sorting a list of this length are much less efficient. Chaining ties (in algorithm S3) does not greatly improve the linear sort, since there are fewer ties than in networks with shorter arc lengths.

The timings reported for network 13 illustrate a point which the authors believe is often missed by those comparing algorithms: the precise list processing techniques can greatly affect the efficiency of the basic algorithm. Therefore we wish to stress that it is not sufficient to describe an algorithm in general terms or give a reference to such a description, when several different list processing techniques are available and the general description is not specific enough to indicate which was actually used. As a dramatic example of this, an algorithm similar to C2, except that the sequence list was treated in a LIFO (rather than FIFO) manner, was run on network 1 and required more than one <u>minute</u> for a single tree, as compared with the slightly more than one <u>second</u> required by C2 to compute the same tree.

Label-setting algorithms cannot be used to compute trees in networks containing negative arc lengths, since they are based on the assumption that nodes can be added to the tree in the order in which they are labeled. If the shortest path from node r to node a goes through node b, the label for the path to b must then be less than the label for the path to a. This may not be the case if negative arcs exist. The following network illustrates this point.



The label-setting algorithms start by examining the forward star of the root node 1, reaching both nodes 2 and 3 from node 1. Since all arcs beginning in the tree also end in the tree, the algorithms terminate at this stage. The path 1 to 2 to 3 is never examined for possible inclusion in the tree. Label-correcting algorithms, on the other hand, will examine this path, since they terminate only when no further path improvements are possible.

The number of computer storage locations required varies for each of the seven algorithms. If N is the number of nodes and M the number of arcs in the network, all algorithms require 3N locations for the arrays a, p, and d, and 2M locations for the arrays n and ℓ. (These arrays will be described in the next section.) The seven algorithms all require additional storage which is summarized in Table 2, where "node arrays" are arrays requiring one full location for each node, "Boolean node arrays" require one bit for each node, and "max arc length arrays" require a number of full storage locations equal to the maximum arc length. Thus algorithm C1 requires the least amount of storage and is also comparatively fast except on the long narrow networks which none of the label-correcting algorithms performed well on. C2 requires only one additional list and is consistently better than C1. It is particularly recommended for networks in which the arcs are all

TABLE 2

ADDITIONAL COMPUTER STORAGE REQUIREMENTS FOR

SHORTEST PATH ALGORITHMS

ALGORITHM	NODE ARRAYS	BOOLEAN NODE ARRAYS	MAX ARC LENGTH ARRAYS
C1		1	
C2	1	1	
C3	2	2	
S1	2		
S2	2		
S3	3	1	
S4	2	1	1

approximately the same length, so that the shortest path between two nodes is very likely to be the path having fewest arcs.

In conclusion, algorithm S4 is recommended whenever it is feasible to set up the distance list which it requires. For networks whose arc lengths do not permit setting up such a list, one of the two label-correcting methods C1 or C2 is recommended, depending on the amount of computer storage available.

8. FLOWCHART CONVENTIONS

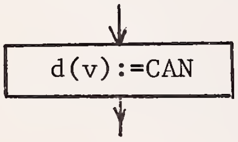
This section will describe several conventions adopted for the flowcharts in section 9. The operations used in shortest path labeling algorithms consist, for the most part, of putting information into or extracting information from various lists. The information to be handled takes three possible forms:

1) integer numbers,
2) the Boolean values, "YES" and "NO", and
3) positions in lists (usually node numbers or arc numbers).

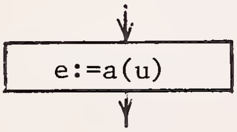
Convention 1: Names of storage locations containing integers or Booleans consist of three capital letters, e.g. TES, AUX, CAN. Single lower case letters will be used to denote storage locations containing list positions, e.g. v,w,e.

Convention 2: List functions whose values are integers or list positions will be denoted by single lower case letters, e.g. d(v), p(w), a(u). List functions whose values are Booleans will be denoted by single capital letters, e.g. C(w), T(u), H(v).

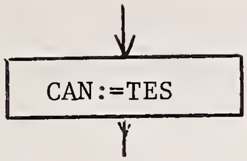
Convention 3: Rectangular boxes will be used for operations which transfer information from one location to another and also for arithmetic operations. (The only arithmetic operations involved in the shortest path algorithms are addition, subtraction, multiplication, and division. These last two are required only in algorithm S2.) Examples of such operations are:



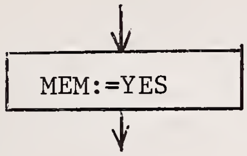
This puts the value of CAN into position v of the list function d .



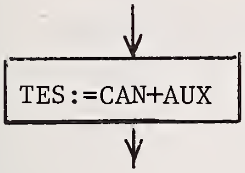
This extracts the value in list position u of list function a and puts it into location e .



This shifts the value in TES into location CAN.

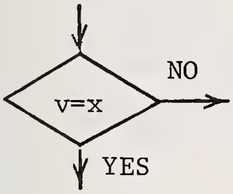


This makes the Boolean variable MEM have value YES.

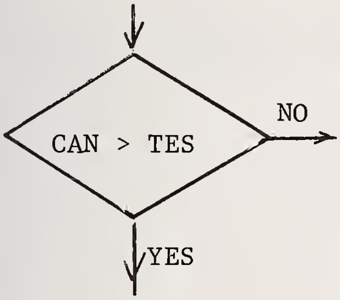


This puts the sum of the variables CAN and AUX into the location TES.

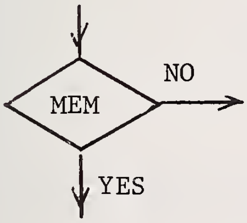
Convention 4: Diamond-shaped boxes will be used for conditional transfer operations. Examples of these operations are:



Transfer on equality. If v=x continue in the YES direction, otherwise in the NO direction.

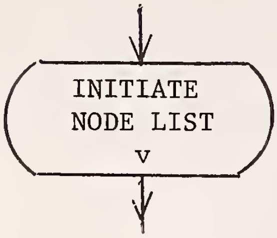


Transfer on comparison. If the value in CAN is greater than that in TES continue in the YES direction, otherwise in the NO direction.

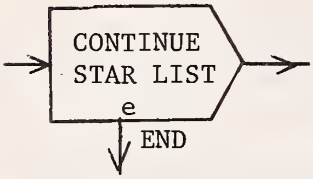


Boolean transfer. If the Boolean variable MEM has value YES continue in that direction, if NO in the NO direction.

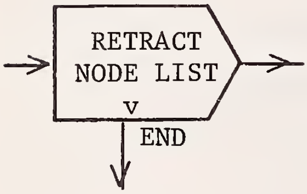
Convention 5: The list operations of initiating, continuing, or retracting a list pointer will be denoted by the following types of boxes.



This loads v with node number 1, the first position in the node list.

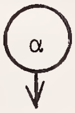


This loads e with the position following the one previously contained in e . The END exit is used when the previous position was the last in the list.

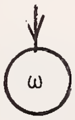


This loads v with the position before the one previously contained in v . The END exit is used when the previous position of v was node 1, the top position in the list.

Convention 6: Circular disks will be used to denote locations in the code. In particular:



denotes the beginning, and



denotes the end of each algorithm.

Convention 7: The following lists and variables will be used in all algorithms.

$a(v)$ denotes the mapping from the node list to the top entry of the star list for each node. The value of a is usually an arc, but it may also be the symbol EMPTY if there is no star list for node v .

$n(e)$ denotes the mapping from the arc list to the node list giving the terminal node of arc e .

$\ell(e)$ denotes the length of arc e .

$d(v)$ denotes the label for node v .

$p(v)$ denotes the node which is the predecessor to node v in the shortest path tree.

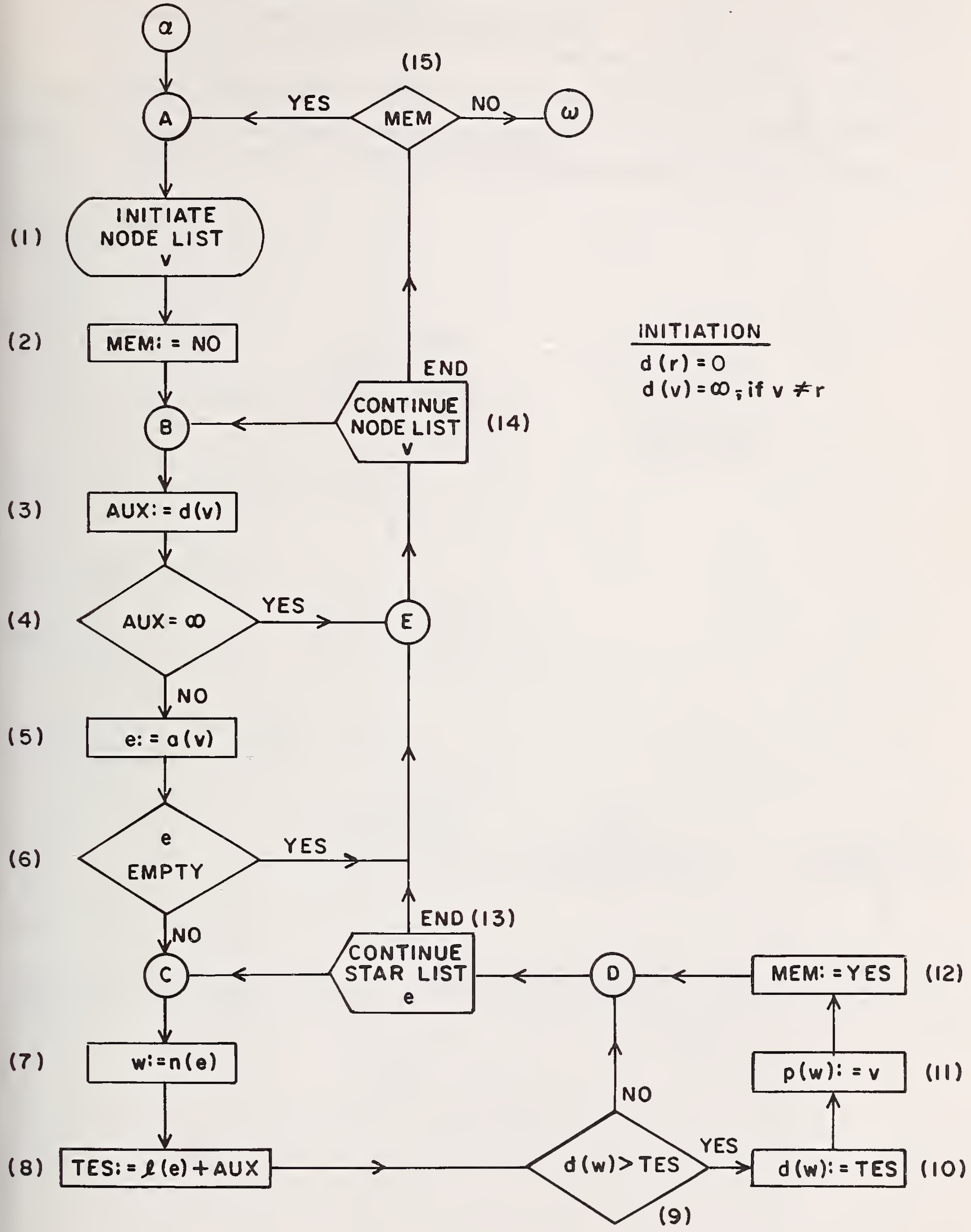
r contains the position of the root in the node list.

9. ALGORITHMS

This section contains descriptions and flowcharts of the basic label-correcting and label-setting methods, and the variations of these basic algorithms discussed in sections 5 and 6. FORTRAN computer programs implementing each of these algorithms appear as an appendix.

C: basic label-correcting. This algorithm realizes the basic label-correcting method given in section 4. The algorithm starts with the root having label zero and all other nodes having infinite label. The node list is searched in order (boxes 1 and 14) at each stage. Forward stars of all nodes with finite labels are examined (boxes 3 through 6, and 13) for arcs e which can be adjoined to the tree to diminish a label (boxes 7 through 9). Once such an arc is found the label and tree are updated (boxes 10 and 11). The Boolean variable MEM is used to indicate whether, during the search through the node list, any labels have been changed (box 12). Termination thus occurs when MEM remains NO throughout a complete search of the node list (box 15).

C: BASIC LABEL CORRECTING



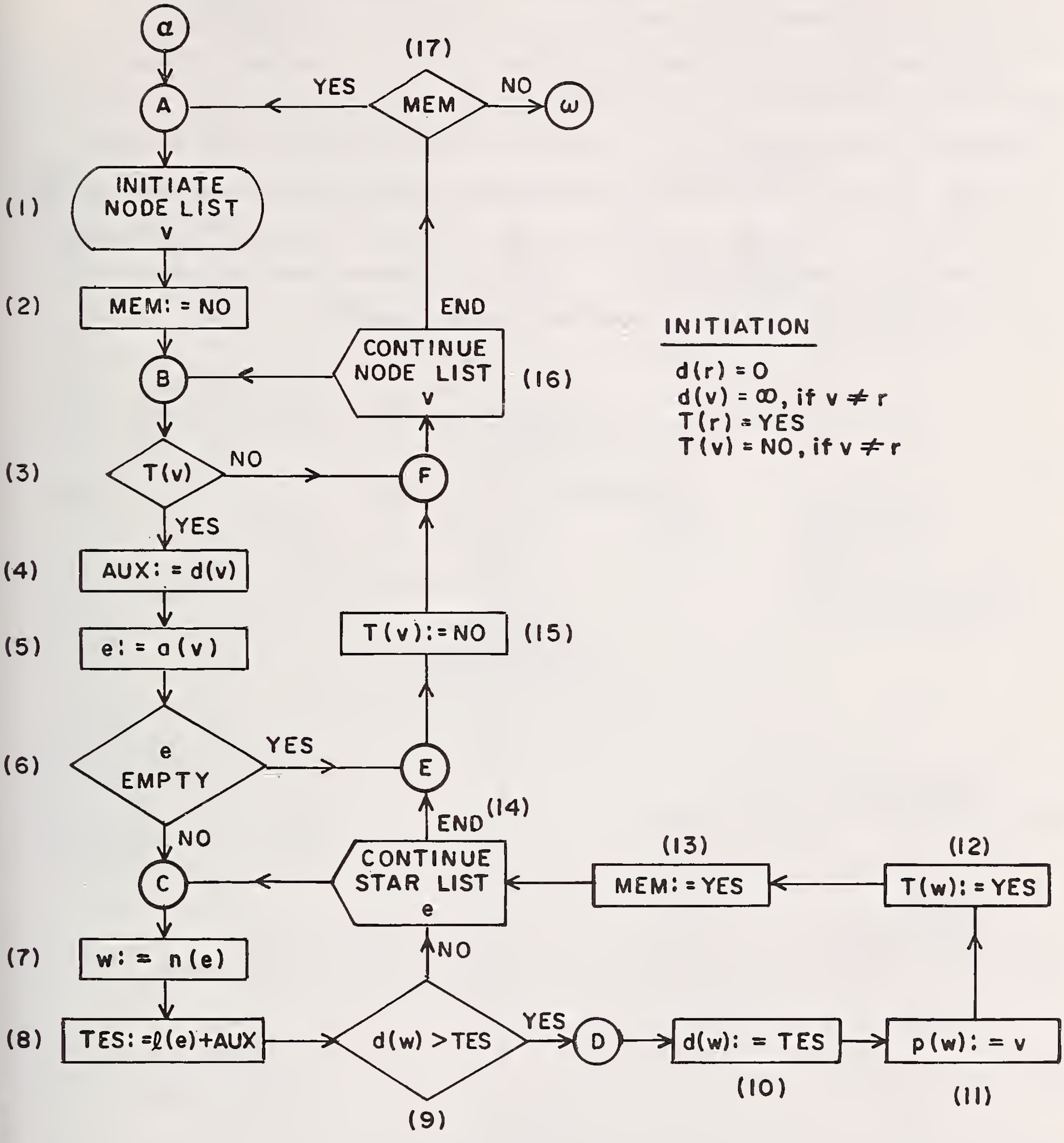
INITIATION

$d(r) = 0$

$d(v) = \infty$; if $v \neq r$

C1: alteration flag. This algorithm utilizes an alteration flag T as described in section 5. The flag T(v) has initial value NO for all nodes except the root r . It is set to YES only when the label of v is changed (box 12) and remains YES until the forward star of v is examined (box 15). The rest of the algorithm is the same as algorithm C , except that it is not necessary to test whether AUX is infinite since T(v) is YES only for nodes with finite labels.

C1: ALTERATION FLAG

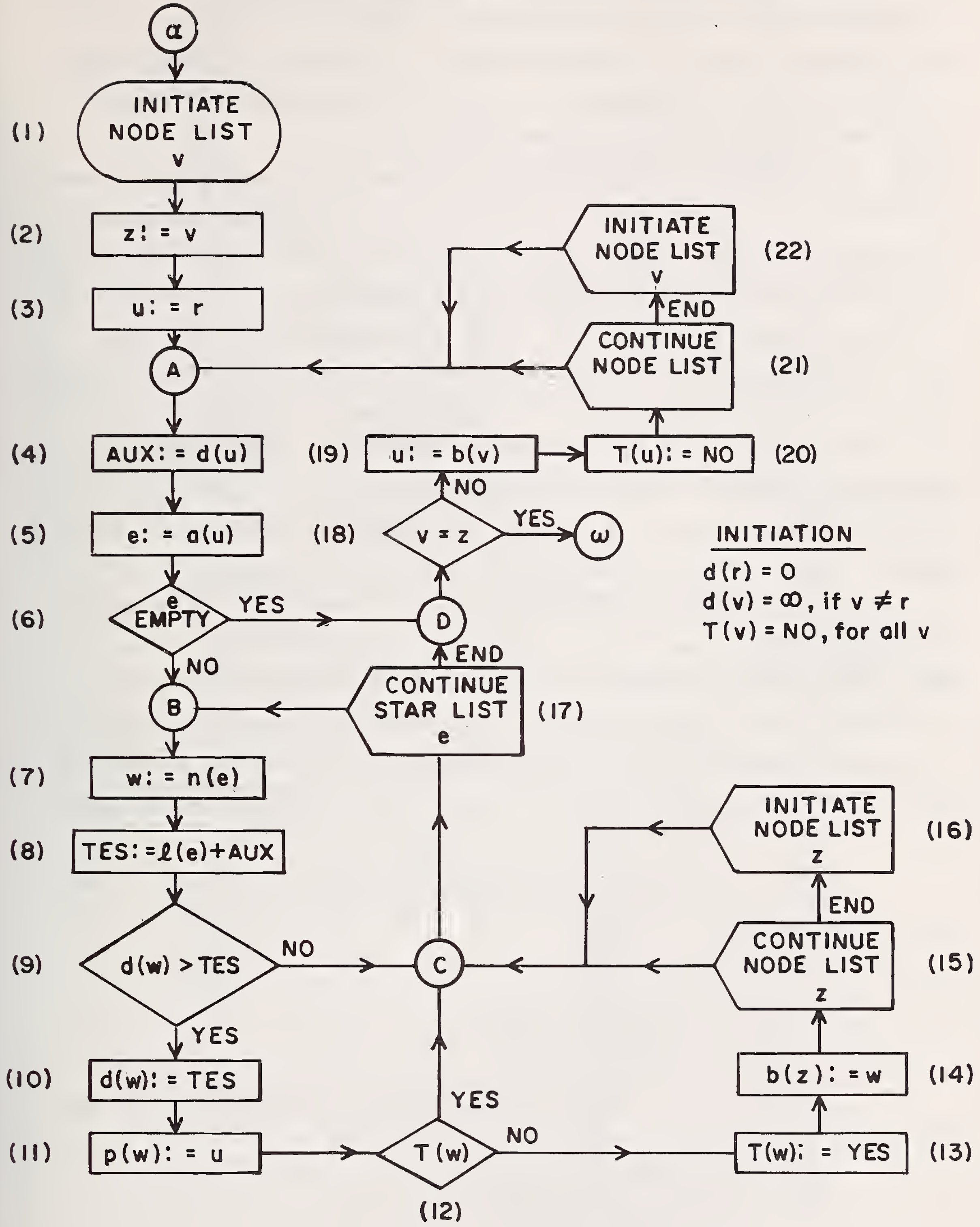


INITIATION

$d(r) = 0$
$d(v) = \infty$, if $v \neq r$
$T(r) =$ YES
$T(v) =$ NO, if $v \neq r$

C2: FIFO sequence list, single entry. This algorithm uses a sequence list b which is treated in a FIFO manner. Nodes are added to the bottom of the list (box 14), indicated by the pointer z as their labels are changed, and nodes are removed from the current top of the list (box 19), indicated by the pointer v . The list is treated in a rotating manner so that whenever either v or z reaches the end of the list, it continues at the beginning (boxes 15 and 16, and 21 and 22). Termination occurs when v = z (box 18), indicating the list is empty. The flag T is used to insure that no node appears more than once on the list (box 12). T(w) is initially set at NO and is changed to YES when w is added to the list (box 13). Once the forward star of a node u has been examined, T(u) is reset back to NO (box 20).

C2: FIFO SEQUENCE LIST, SINGLE ENTRY



INITIATION

$d(r) = 0$

$d(v) = \infty$, if $v \neq r$

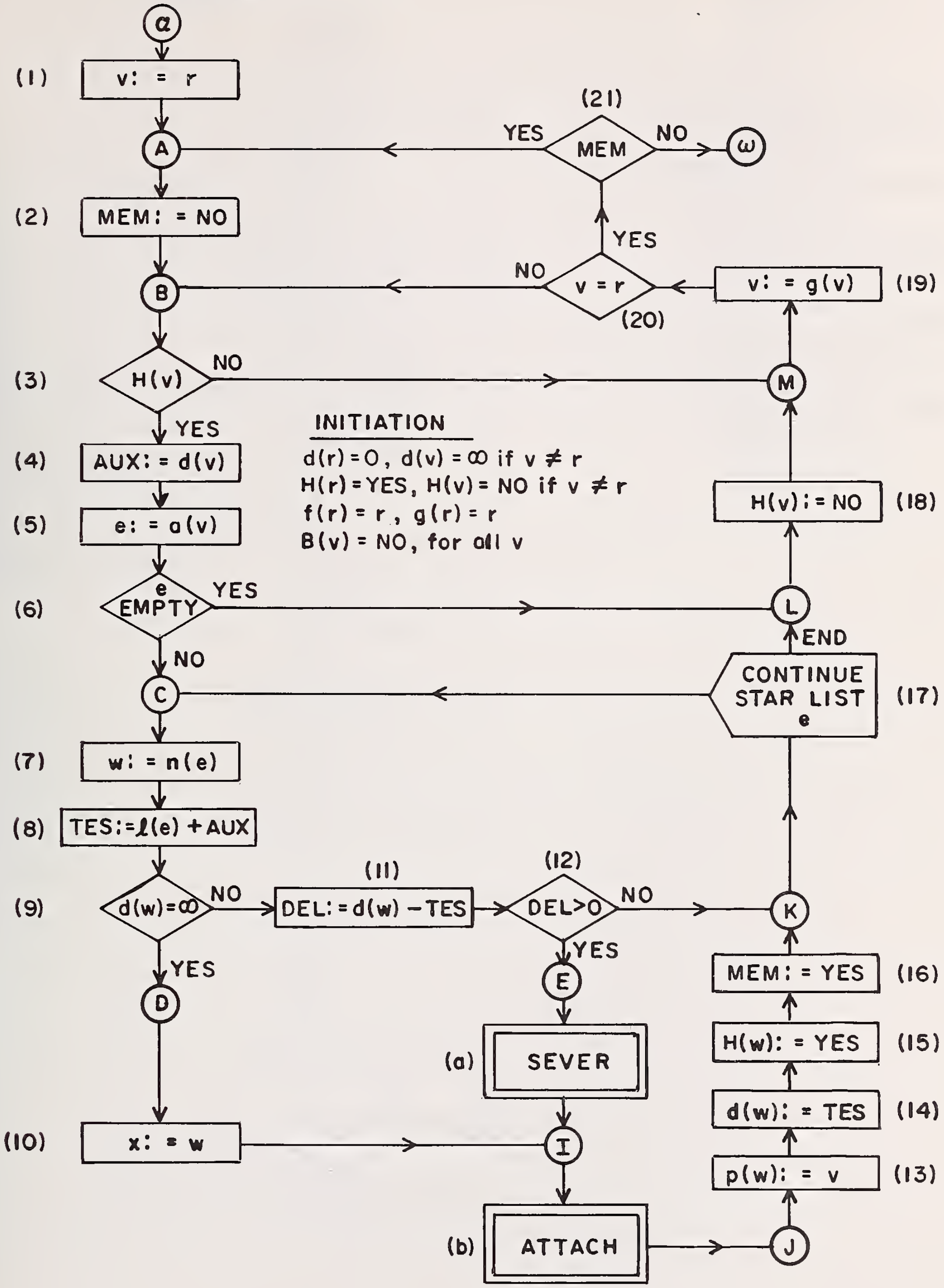
$T(v) = NO$, for all v

C3: forward pointer, sequencing by inverse of the forward pointer. This algorithm utilizes a forward pointer f as described in section 5 to implement the process of severing and reattaching branches. The algorithm also uses an alteration flag H. Sequencing of nodes occurs through the use of the inverse g of the forward pointer (box 19); g being such that if $f(u) = v$ then $g(v) = u$.

SEVER. This section of algorithm C3 decreases the labels on the branch attached to w by the amount DEL (box 107), at the same time setting the flags on all the nodes in this branch (box 108). Boxes 101 through 106 and 109 find the nodes in the branch attached to w . These nodes x are indicated by $B(x) = \text{YES}$. This is accomplished by setting $B(w) = \text{YES}$ and following the forward pointer from w until either, for some node v the predecessor of v is not in the branch attached to w , or some node v in the branch has $f(v) = r$. Boxes 110 through 112 sever the branch from the old predecessor of w , updating both the forward pointer f and its inverse g ; boxes 113 through 116 reset B

ATTACH. This section of algorithm C3 attaches the branch just severed to the node v , the new predecessor of w . Again this involves updating both the f and the g pointers. After SEVER, x is the last node in the branch attached to w . If there is no branch attached to w , then x is w (box 10 of C3). Otherwise the branch is attached to v by inserting it in the forward pointer list between v and the old f(v). Thus f(v) is set equal to w and f(x) is made the old f(v).

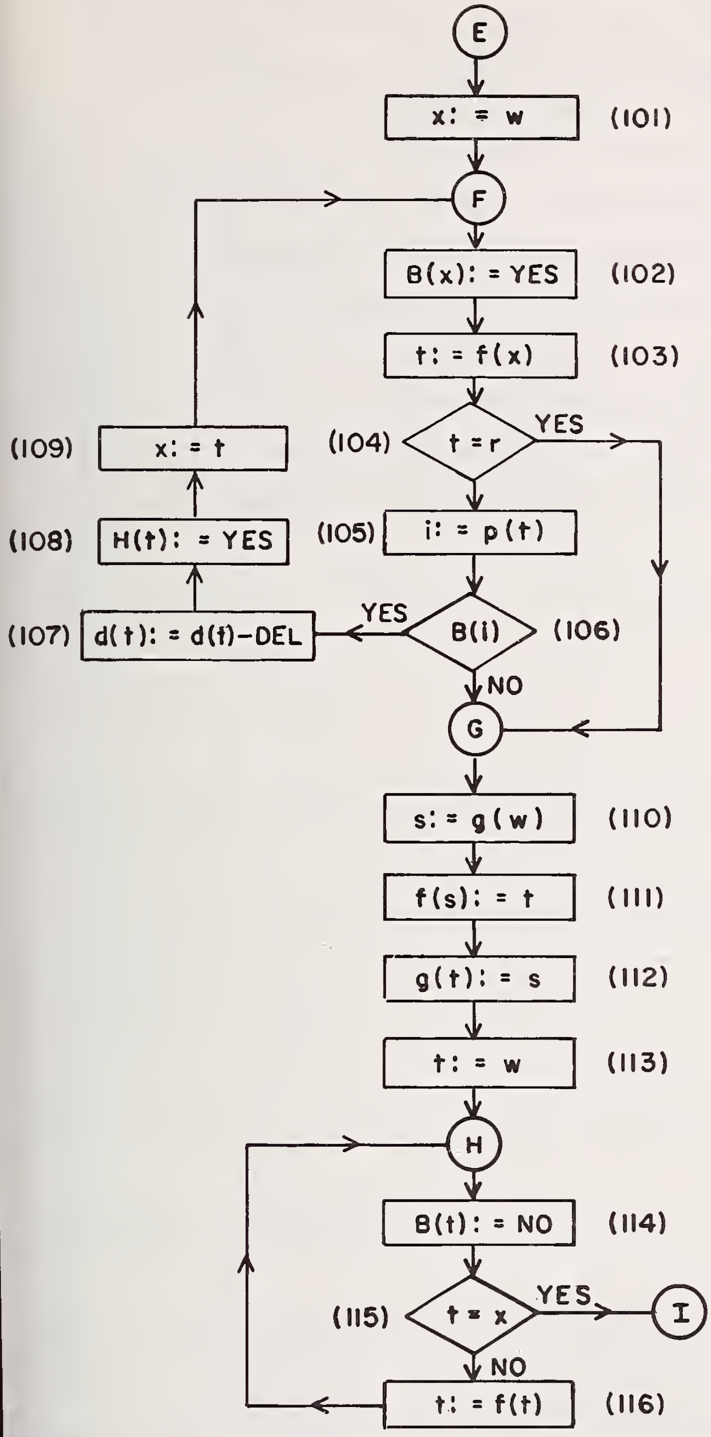
C3: FORWARD POINTER, SEQUENCING BY INVERSE OF THE FORWARD POINTER



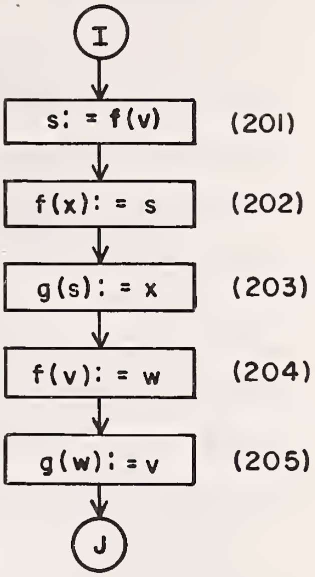
INITIATION

d(r) = 0, d(v) = ∞ if v ≠ r
H(r) = YES, H(v) = NO if v ≠ r
f(r) = r, g(r) = r
B(v) = NO, for all v

SEVER

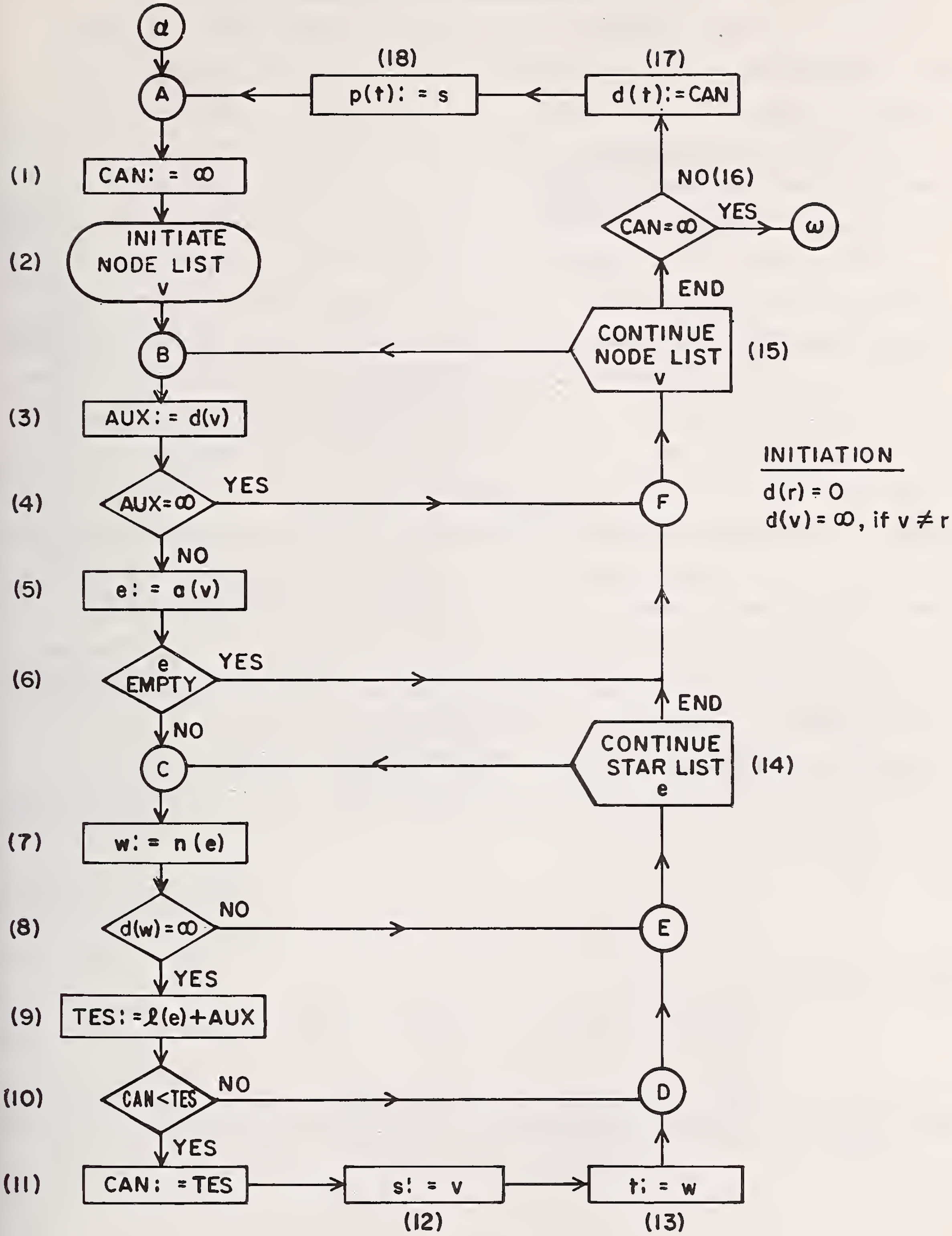


ATTACH



S: basic label-setting. This algorithm realizes the basic label-setting method described in section 4. The algorithm starts with the root labeled zero and all other nodes having infinite labels. During each pass all nodes are examined (boxes 2 and 15). All arcs leading from a node with finite label to a node with infinite label (i.e. from a node in the tree to a node not yet in the tree) are examined as possible extensions (boxes 3 through 9). The current minimum potential label is stored in CAN, the starting node of the new tree arc in s, and its end in t (boxes 10 through 13). CAN is initialized at ∞ at the start of each pass (box 1). The algorithm terminates when no new arcs can be added to the tree, i.e. when CAN is still infinite after all nodes have been examined (box 16). If CAN is finite then the node t is added to the tree and its label is made CAN (boxes 17 and 18).

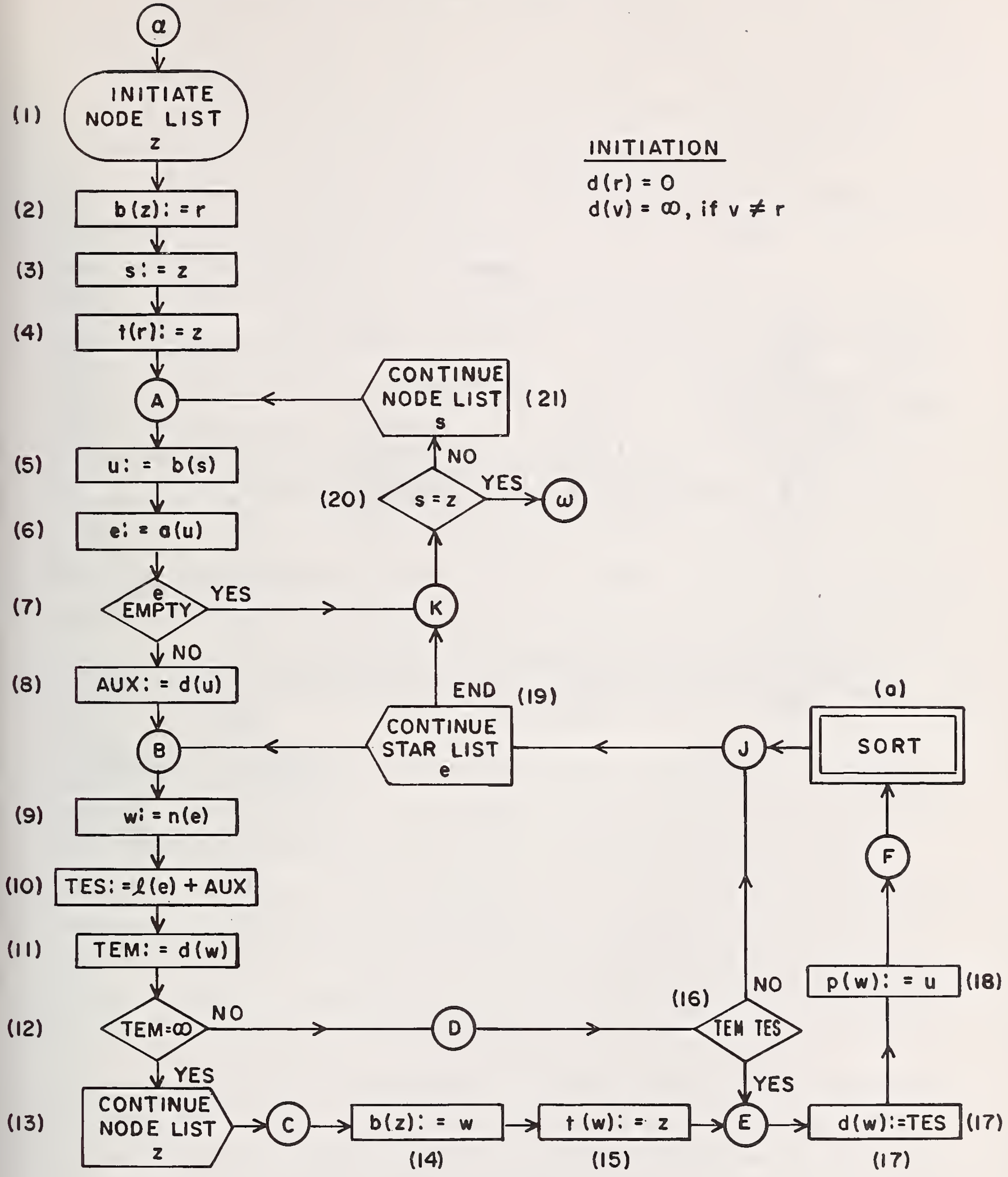
S: BASIC LABEL SETTING



S1: sequencing by label, linear sort. In this algorithm, the first time a node not yet in the tree is encountered (box 12), it is put on the list b which is linearly ordered by the temporary labels (boxes 13 through a). The mapping t is the inverse of b, i.e. if w occupies the kth position in list b , then b(k) = w and t(w) = k . Mapping t is required in order to find the position occupied by a node w already on the list when its label is changed, since w will be moved up in the list from this position by SORT. Two pointers to the list b are used: s , which points to the current top position, and z , which is the current last node in the list. There is no need to make a distinction between temporary and permanent labels,since each node between positions s and z on the sequence list has a temporary label, while those above position s and that in s itself are permanently labeled. Boxes 13, 21, and 103 have no END exit since the end of the list will never be reached.

SORT. This section of algorithm S1 puts w in the correct position in the list b to insure that the list remains linearly sorted by label. The instructions in boxes 14 and 15 put w at the bottom of list b if it is not already on the list (box 12). Variable y starts as the position occupied by w and is then retracted (boxes 101 through 103). If the node v in position y has a larger label than w , v is moved one position down the list. When finally the label of w is greater than or equal to the label of some node v , w is placed in the position below v .

S1: SEQUENCING BY LABEL, LINEAR SORT

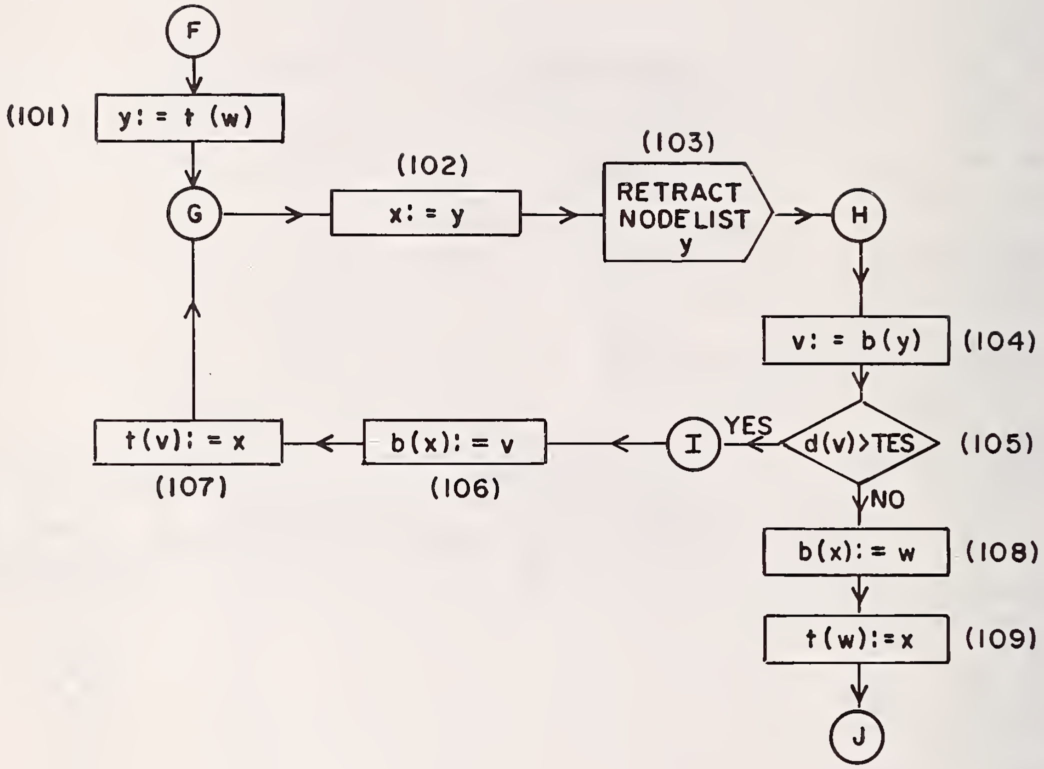


INITIATION

$d(r) = 0$

$d(v) = \infty$, if $v \neq r$

SORT

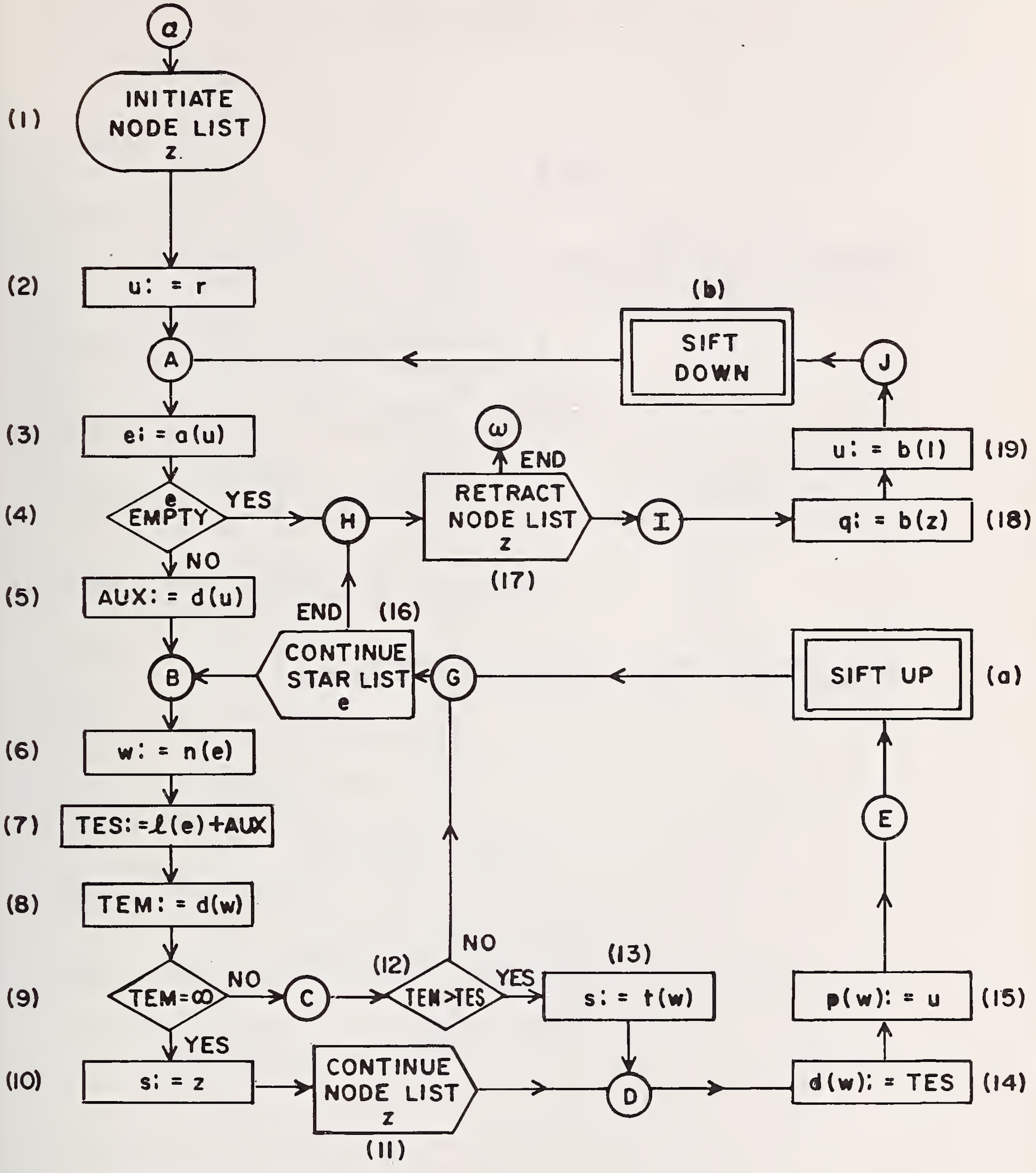

F
(101)
y: = t (w)
G
(102)
x: = y
(103)
RETRACT NODELIST y
H
v: = b(y)
(104)
d(v)>TES
(105)
YES
NO
I
b(x): = v
(106)
t(v): = x
(107)
b(x): = w
(108)
t(w):=x
(109)
J

S2: sequencing by label, tree sort. In this algorithm the sequence list b is maintained in the tree sorted fashion described in section 6: if $i = [j/2]$ (fixed point division) then $d(b(i)) \leq d(b(j))$. Mapping z points to the next entry to be filled at the bottom of the list. Node u , the one whose forward star is examined next, starts out as the root r and from then on is the top node in b (box 19). The algorithm proceeds much as in algorithm S1. The procedure SIFT UP puts w in the correct position in b , and the procedure SIFT DOWN rearranges b to maintain the tree ordering after the top node is removed. The algorithm terminates when the list b is empty.

SIFT UP. This section puts w in a position in list b so that the tree ordering is maintained. Location s contains the trial position for w . If w is not in the list b , then s starts at the end of the list (boxes 9 and 10). If w is already in the list, then s starts at the current position occupied by w (box 13). Variable h is the position of the "predecessor" of s in the tree sort (box 101). If there is no such predecessor (box 102) then w is put at the top of the list. Otherwise the label of w is compared with the label of the node v in position h (boxes 103 through 105), and if the label for w is less, v is moved down to position s (boxes 106 and 107). This process continues until for some value of h , $d(v) \leq d(w)$. Then w is put in position s .

SIFT DOWN. This section reorders the list b after the top node is removed. The last node q is removed from the list b (boxes 17 and 18). Since the top node has also been removed (box 19), there is a free position in the list in which to put q . The SIFT DOWN procedure searches for the correct such position s , moving nodes up the list until it is found. Variable s starts at the top position, which is now empty (box 201). At each stage, h is one of the two positions 2s or 2s+1 for which s is the "predecessor" (boxes 203, 207, and 214). The label of w is compared with the lesser of the labels of the nodes v in position h and y in position h+1 (boxes 203 through 215). If d(w) is greater than the minimum of the two, the node having that minimum label is moved up the list, and the new trial position s becomes the position previously occupied by the node which was moved (boxes 216 through 218). This procedure insures that the new top node has minimum label and that the rest of the list remains tree sorted.

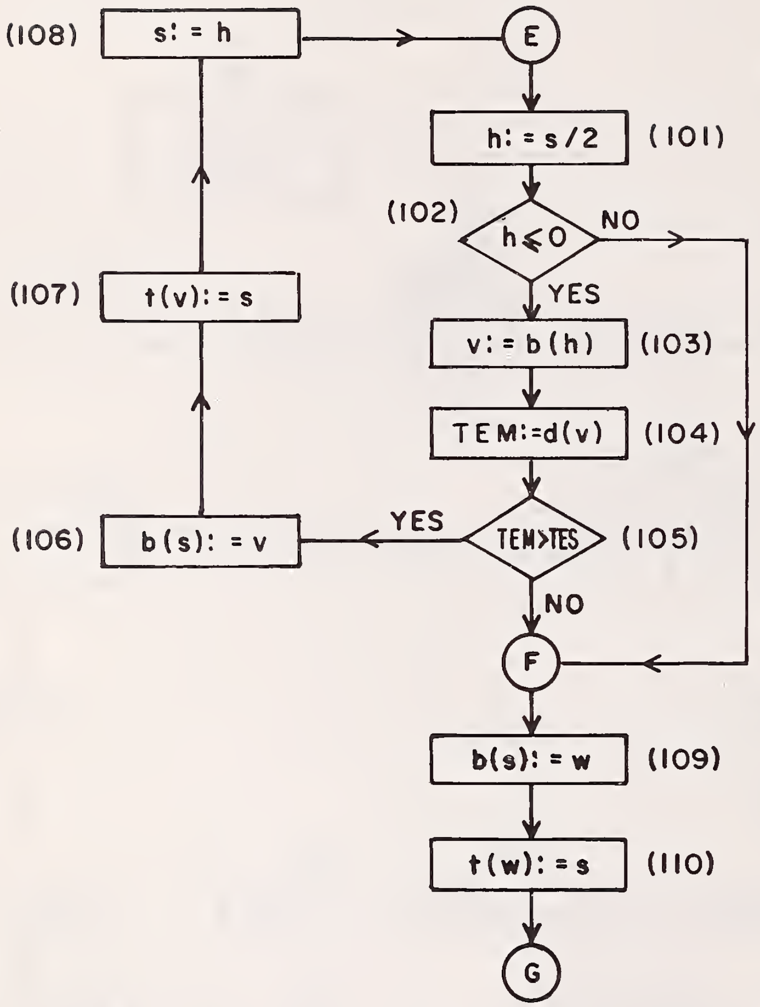
S2: SEQUENCING BY LABEL, TREE SORT



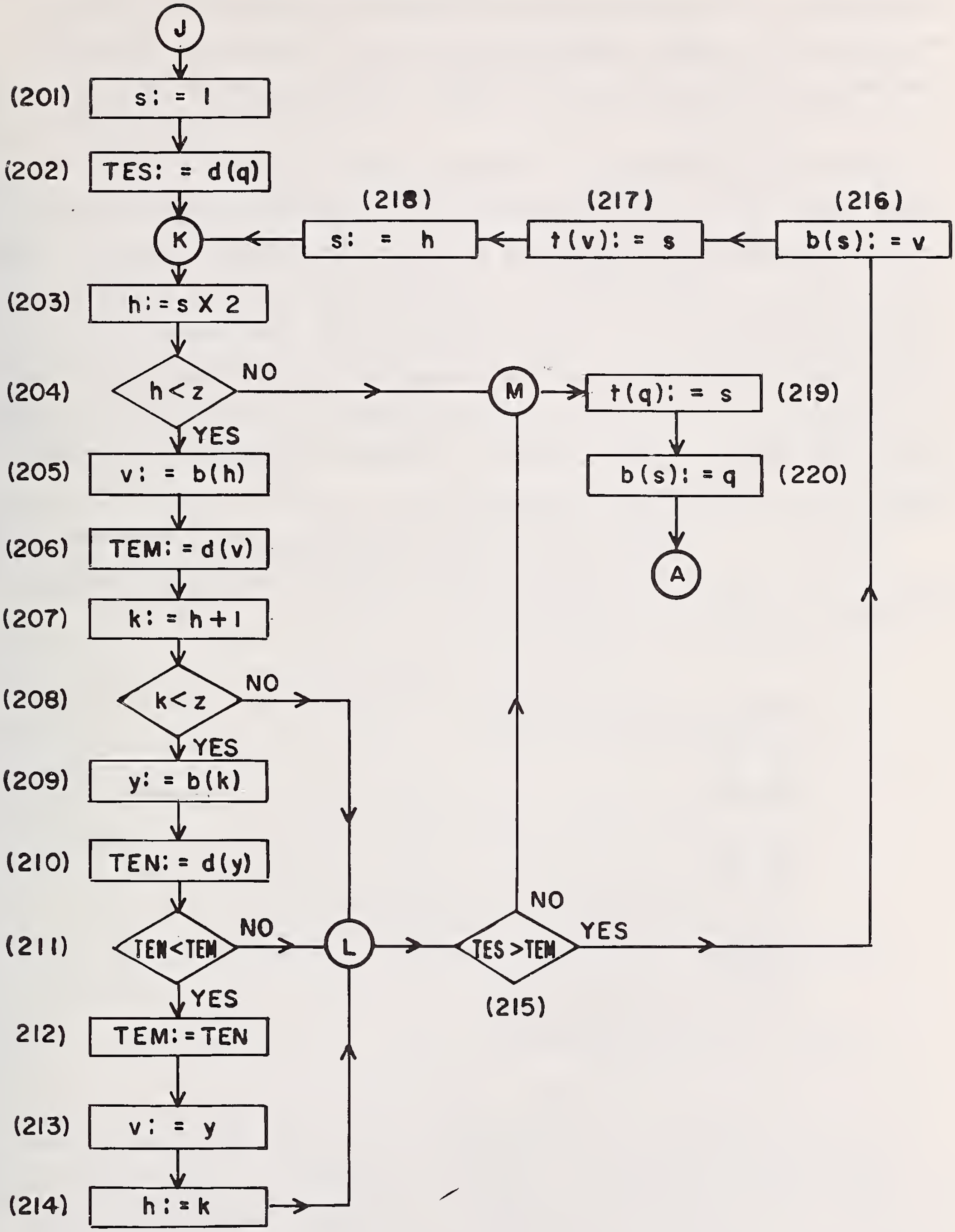
INITIATION: $d(r) = 0$

$d(v) = \infty$, if $v \neq r$

SIFT UP

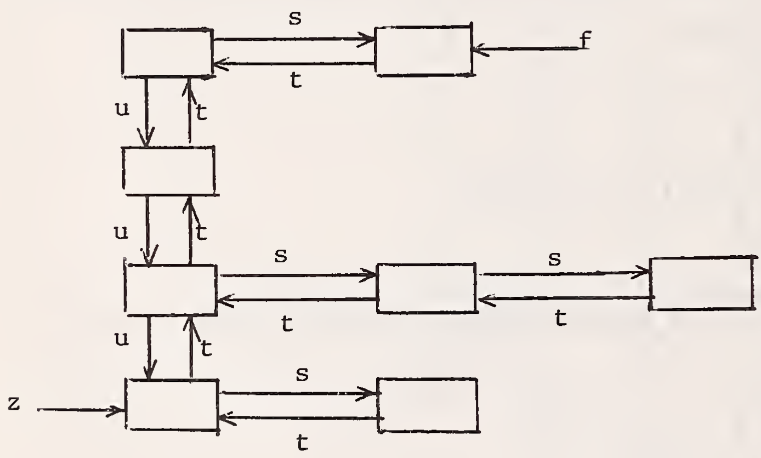

(108)
s: = h
E
h: = s/2
(101)
(102)
h≤0
NO
(107)
t(v): = s
YES
v: = b(h)
(103)
TEM:=d(v)
(104)
(106)
b(s): = v
YES
TEM>TES
(105)
NO
F
b(s): = w
(109)
t(w): = s
(110)
G

SIFT DOWN


J
(201) s: = 1
(202) TES: = d(q)
K
(203) h: = s X 2
(204) h < z
NO
YES
M
(219) t(q): = s
(220) b(s): = q
A
(205) v: = b(h)
(206) TEM: = d(v)
(207) k: = h + 1
(208) k < z
NO
YES
(209) y: = b(k)
(210) TEN: = d(y)
(211) TEN < TEM
NO
YES
L
(215) TES > TEM
NO
YES
212) TEM: = TEN
(213) v: = y
(214) h: = k
(216) b(s): = v
(217) t(v): = s
(218) s: = h

S3: sequencing by label, linear sort with chained ties. In this algorithm the sequence list is maintained in a linear sorted form as in S1, but pointers are used to chain ties (nodes having the same label) in such a manner that searching the list for the correct position for a new node involves only comparing the label of the new node with the label of one of the nodes in any chain. The main part of the algorithm is similar to S1 and S2. Node v is the one whose forward star is currently being examined. If a node w is to be added to the list (box b), then if it is already on the list (box 10), it must first be removed from its current position (box a). Two pointers to the list are used, namely f which points to the next entry whose forward star should be examined, and z which points to the first entry in the bottom chain on the list. Box 19 removes the top element in the list and stores it in v , and f is then advanced to be the new top element in the list (box c). The algorithm terminates when there is no top element (i.e. f=0) after a pass (box 18).

The list is arranged in the following manner, where elements at the top of the list have lower labels than elements below and elements on the same line have the same label.



f is the last element in the top chain and z is the first element in the bottom chain. Mapping u is only defined for elements which are the first of a chain, and it points to the first element of the next lower chain. Therefore u(z) = 0 . Pointer s indicates the next element in the same chain; s is zero for the last element in any chain. If node v is the first element in a chain, then t(v) points to the first element in the chain immediately above v , and t(v) = 0 if and only if v is the first element in the top chain. For elements within a chain, t points to the preceding chain element. The Boolean flag H is used to indicate which nodes are first elements ("heads") of some chain.

REMOVE w . This section of the algorithm is entered whenever a new lower label is found for a node w which is already on the list. Node w must be removed from the position it formerly occupied before it can be put in a new position. If w is not the first element of a chain, then it is necessary only to update the s and t pointers (boxes 103 and 120 through 122). If w is the first and only element of a chain, then removing w consists of updating only the t and u pointers (boxes 105 and 115 through 119). If w is the first element of a chain which contains other elements then s(w) becomes the new head of the chain and the t and u pointers are updated accordingly.

When this section is exited, several conditions hold:

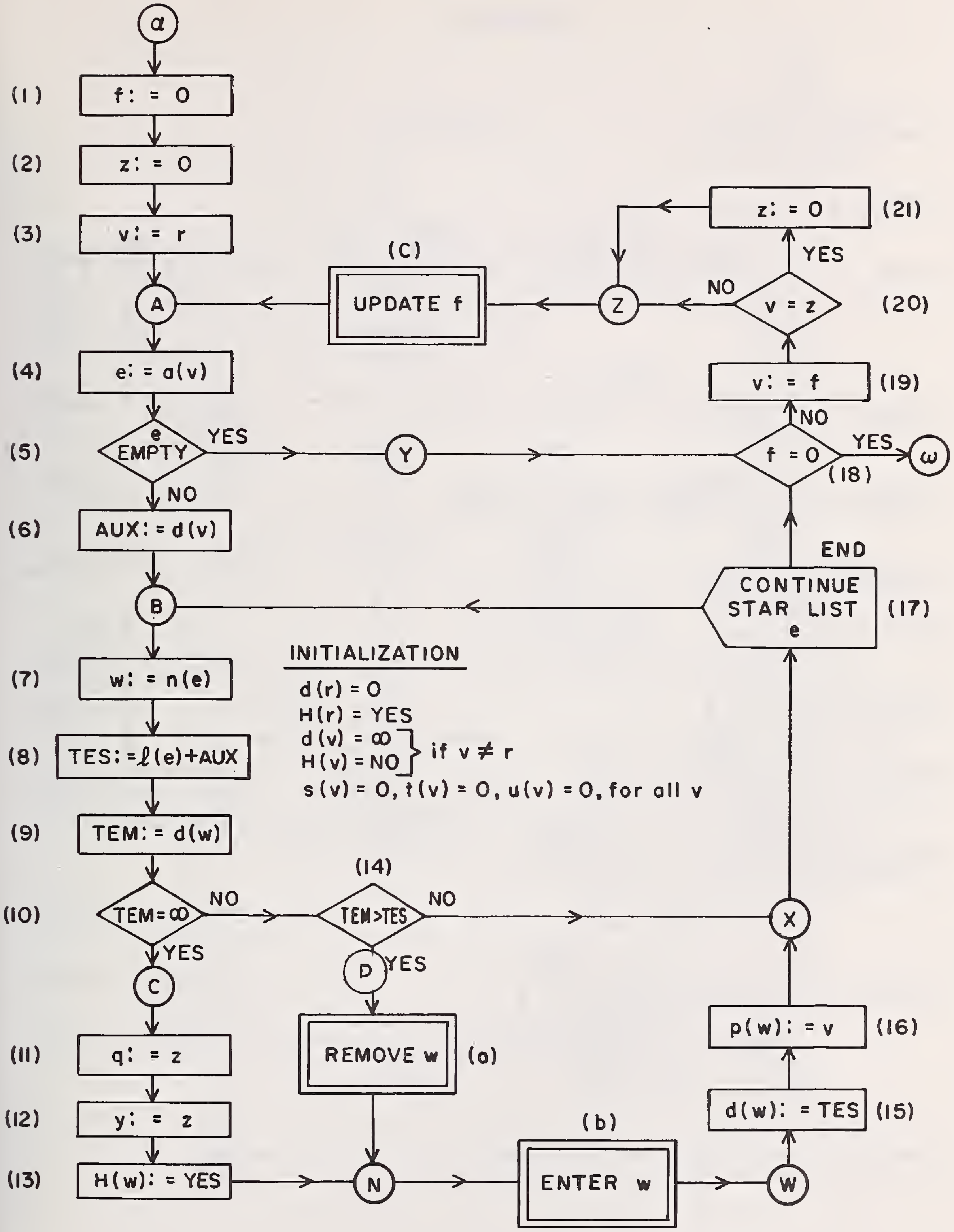
1) the t, s, and u pointers have been updated in such a manner that w is no longer in the list,
2) H(w) = YES, since ENTER w will put w at the head of some chain,
3) q contains the first element of the chain immediately above that formerly occupied by w , and
4) y contains the first element of the chain below the chain containing q .

These last three properties are assured in boxes 114, and 123 through 126 , whenever not done in boxes 102 or 104.

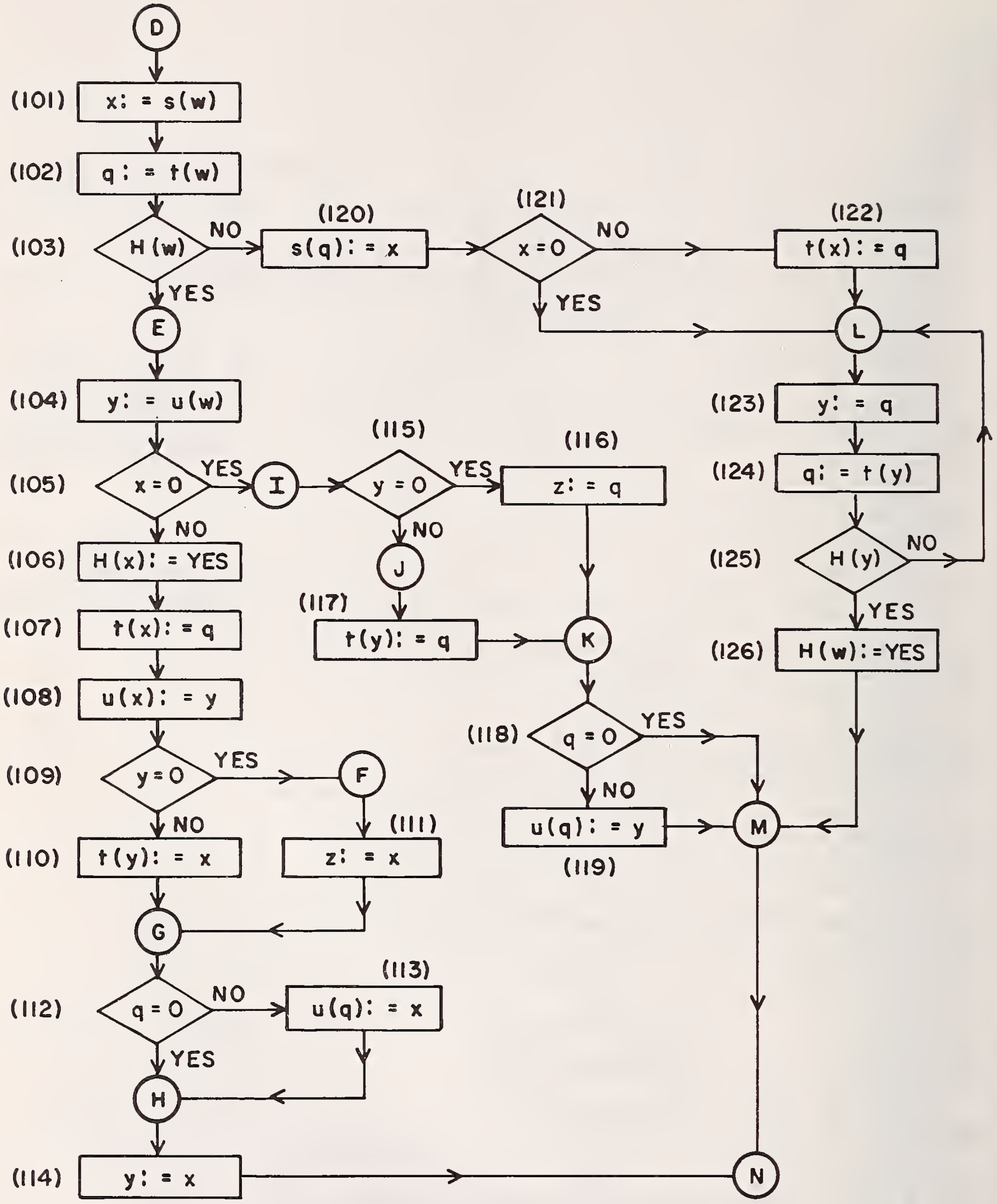
ENTER w . This section of the algorithm finds the correct chain for w and puts w at the head of that chain. When this section is entered, q, the current trial position for w , is either z , the end of the list (box 11), or was set in REMOVE w as the head of the chain directly above that previously containing w . If q = 0 , then q is the top of the list and f is updated to be w (boxes 201 and 218). Otherwise the label of the head of each chain is compared with TES, the label of w , until for some q its label is less than or equal to TES (boxes 201 through 206). If the label of q is less than TES, then w becomes the head of a chain containing only itself (boxes 203 and 219 through 225). If q has the same label as w , then w becomes the new head of the chain containing q (boxes 204 and 207 through 217).

UPDATE f. When the node f is removed from the list (and stored in v), it is necessary to find a new f . If the chain containing v had several nodes, then the new f is the node preceding v in that chain (boxes 301 and 302). If v was the only node in the chain, then the new f becomes the last node in the next lower chain. This is accomplished by proceeding down by u (box 303) and across by s (boxes 306 and 307), until the last node in the chain headed by u(v) is found. Box 305 insures that if v headed a chain, the t pointer for the first element of the chain below v is set to zero making that chain the new top chain.

S3: SEQUENCE BY LABEL, LINEAR SORT WITH CHAINED TIES

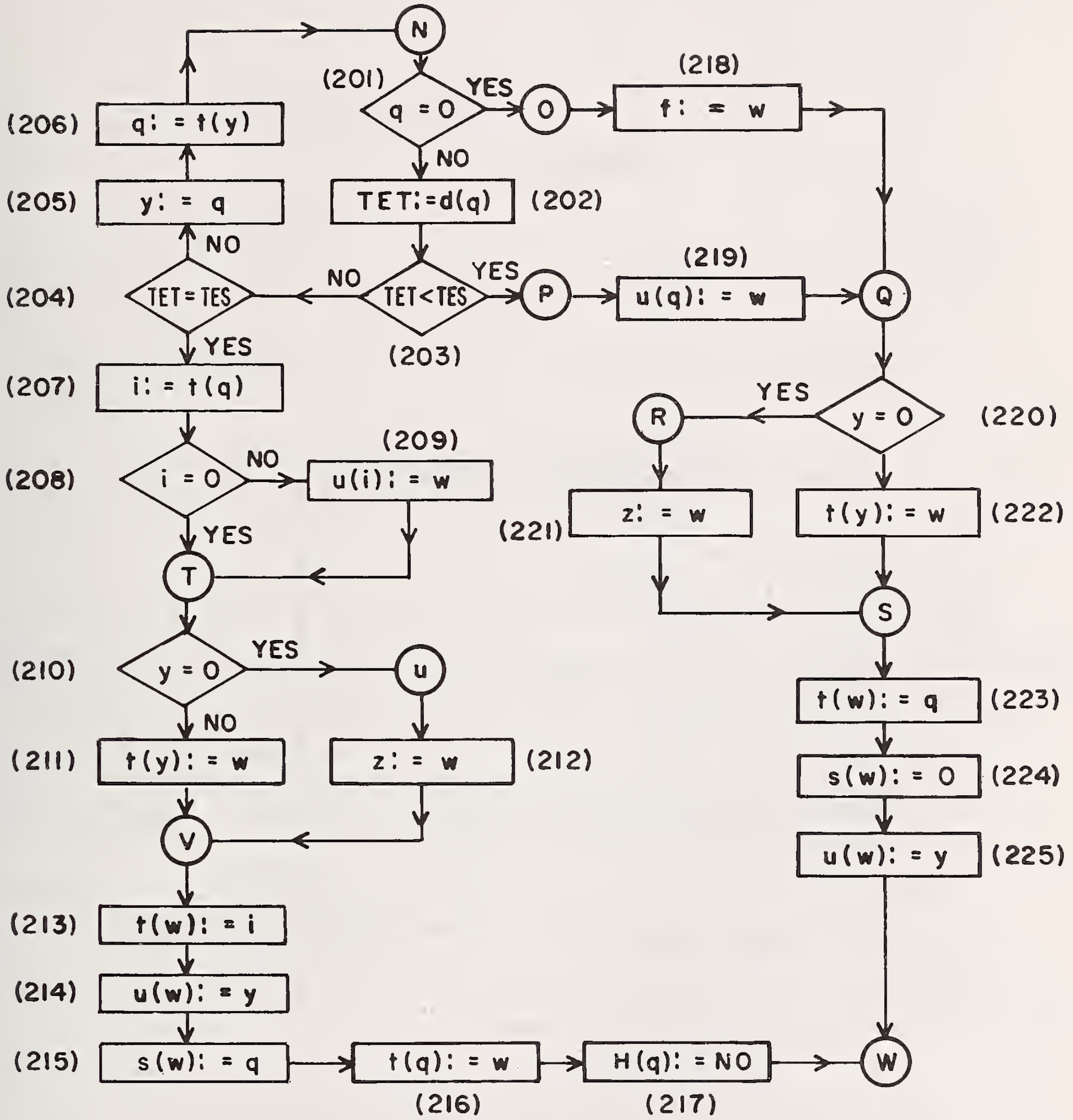


REMOVE w



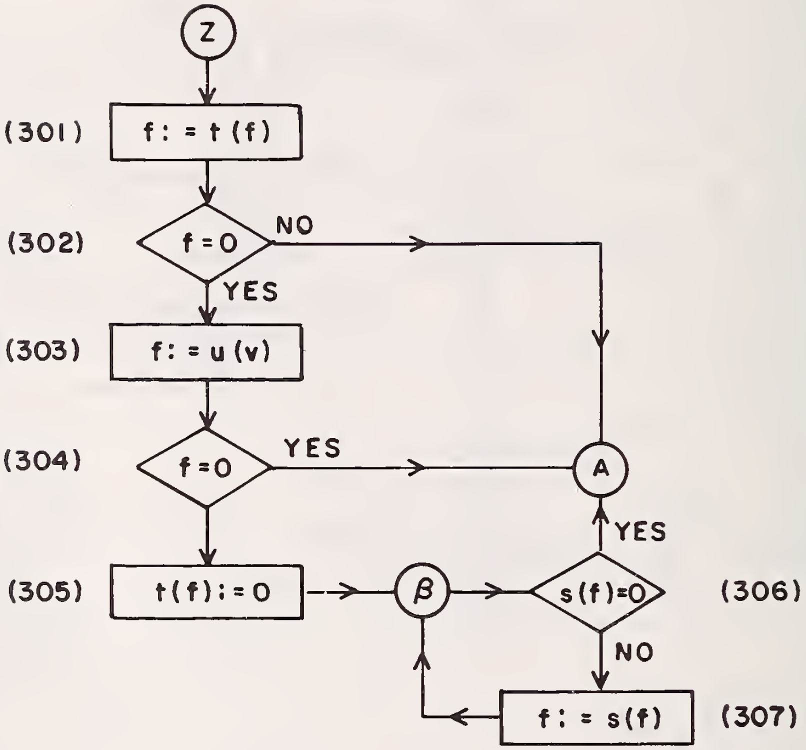
D
(101) x: = s(w)
(102) q: = t(w)
(103) H(w)
NO
YES
E
(120) s(q): = x
(121) x = 0
NO
YES
(122) t(x): = q
L
(104) y: = u(w)
(105) x = 0
YES
NO
I
(115) y = 0
YES
NO
J
(116) z: = q
(123) y: = q
(124) q: = t(y)
(125) H(y)
NO
YES
(126) H(w): = YES
(106) H(x): = YES
(107) t(x): = q
(117) t(y): = q
K
(108) u(x): = y
(118) q = 0
YES
NO
(119) u(q): = y
M
(109) y = 0
YES
NO
F
(110) t(y): = x
(111) z: = x
G
(112) q = 0
NO
YES
(113) u(q): = x
H
(114) y: = x
N

ENTER w

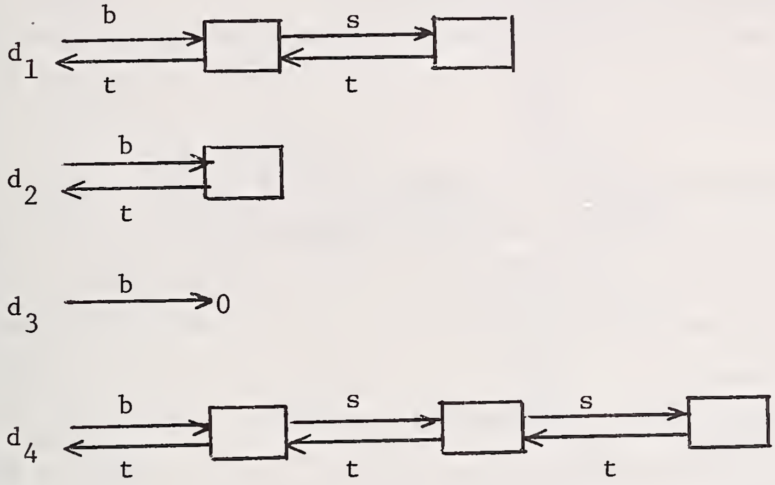


N
(201)
q = 0
YES
O
(218)
f: = w
NO
(202)
TET:=d(q)
(203)
TET<TES
YES
P
(219)
u(q): = w
Q
NO
(204)
TET=TES
NO
(205)
y: = q
(206)
q: = t(y)
YES
(207)
i: = t(q)
(208)
i = 0
NO
(209)
u(i): = w
YES
T
(210)
y = 0
YES
u
NO
(211)
t(y): = w
(212)
z: = w
V
(213)
t(w): = i
(214)
u(w): = y
(215)
s(w): = q
(216)
t(q): = w
(217)
H(q): = NO
W
(220)
y = 0
YES
R
(221)
z: = w
(222)
t(y): = w
S
(223)
t(w): = q
(224)
s(w): = 0
(225)
u(w): = y

UPDATE f


Z
(301)
f: = t (f)
(302)
f = 0
NO
YES
(303)
f: = u (v)
(304)
f = 0
YES
A
YES
(305)
t(f): = 0
β
s(f)=0
(306)
NO
f: = s(f)
(307)

S4: sequencing by label, distance list. In this algorithm a node is placed on the sequence list in a position determined directly by the value of its label. This algorithm requires one additional piece of information about the network: MAX, the maximum arc length plus 1. The sequence list b, which will be called the distance list, has length MAX. Nodes with equal labels are chained on the sequence list in the following manner:



where d_i is a label modulo MAX. Rather than actually using modulo arithmetic, the algorithm keeps track in LEV of the current minimum label as a multiple of MAX. LEV is incremented by MAX whenever the "read" pointer z reaches the bottom of the list (boxes 19 and 22). As for pointer b, $b(y)$ is the first node of the chain of nodes having label $LEV + y$, if $y \geq z$. Otherwise the node has label $LEV + MAX + y$. A flag $H(v)$ is used to indicate those nodes v which head chains. Mapping s points to sucessive nodes in a single chain. For nodes v which head chains, $t(v)$ is the position in the distance list occupied by v, and for nodes v which do not head chains $t(v)$ is the node preceding v in its chain.

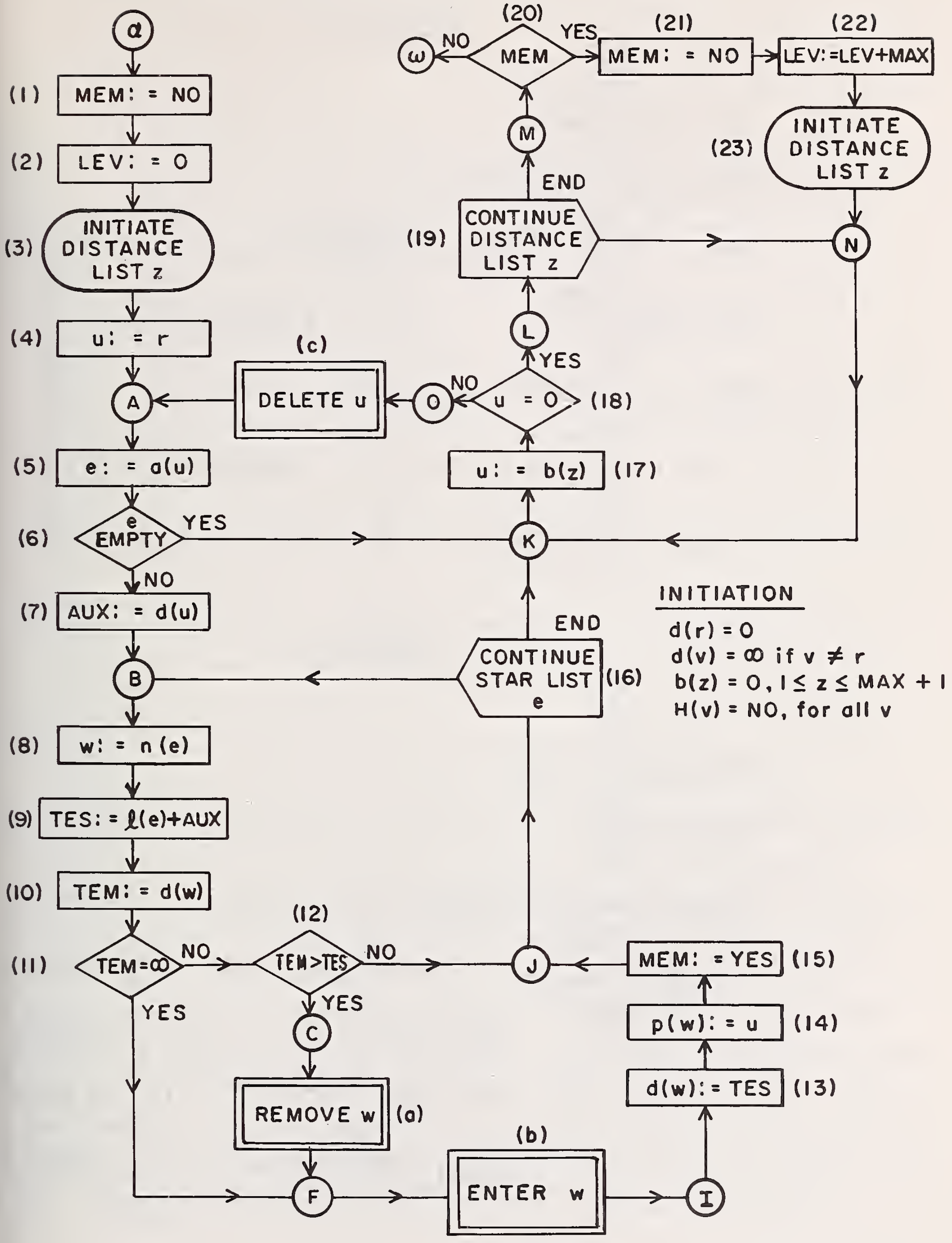
S4 proceeds in much the same manner as S1, S2, and S3, Node u is the one whose forward star is currently being examined (boxes 4 and 17); it occupies position z in the distance list. If a node w which is already in the list is encountered, it must be removed from its present position in the list (boxes 11, 12, and a) before it can be added in its new position. The algorithm terminates whenever, after a complete pass through the distance list, no new nodes have been added to the tree (box 20). The Boolean variable MEM is used to indicate this condition.

REMOVE w . This section of the algorithm removes w from its current position in the list, prior to inserting it in a new position. If w does not head a chain, then deleting w involves only updating the s and t pointers (boxes 103 through 105 and 109). If w heads a chain then b must be updated, and if there are other elements in the chain then the old successor of w becomes the new head of the chain (boxes 106 through 109).

ENTER w . This section enters w in the correct chain in the list. Boxes 201 through 203 determine the correct position y in the distance list, y = TES (mod MAX). Boxes 204 through 211 update the b, s, and t pointers and the Boolean flag H to put w at the head of the chain of nodes in position y .

DELETE u . This section of the algorithm deletes u from the top of the list. Since u was head of a chain (box 17), if it was the only node in the chain, then deleting it involves setting b(z): = 0 for the position z occupied by u (boxes 301 and 302). However, if there were other nodes in the chain containing u , then the first of them, s(u), must be made head of the chain (boxes 304 and 305).

S4: SEQUENCING BY LABEL, DISTANCE LIST



INITIATION

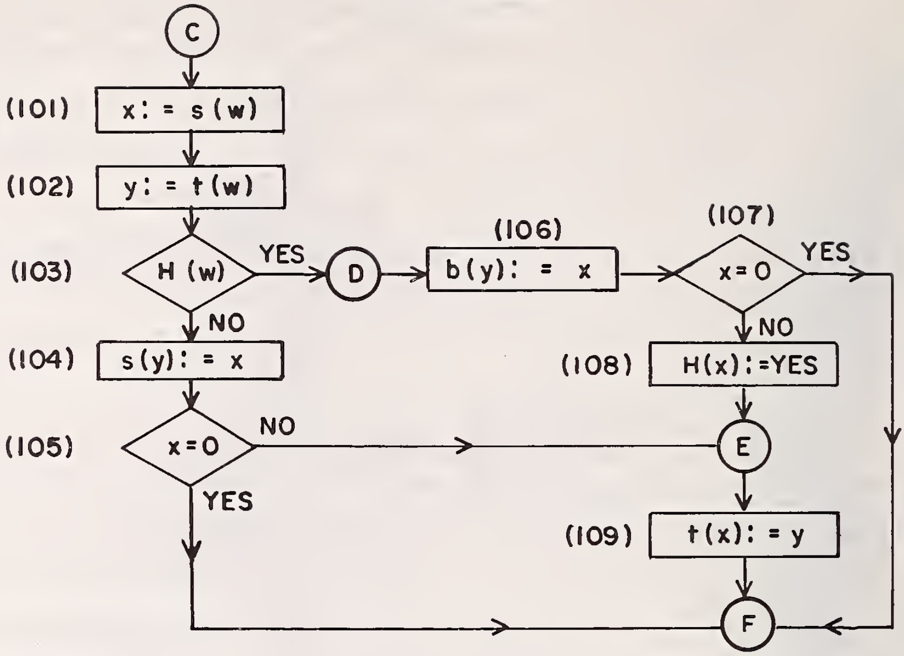
$d(r) = 0$

$d(v) = \infty$ if $v \neq r$

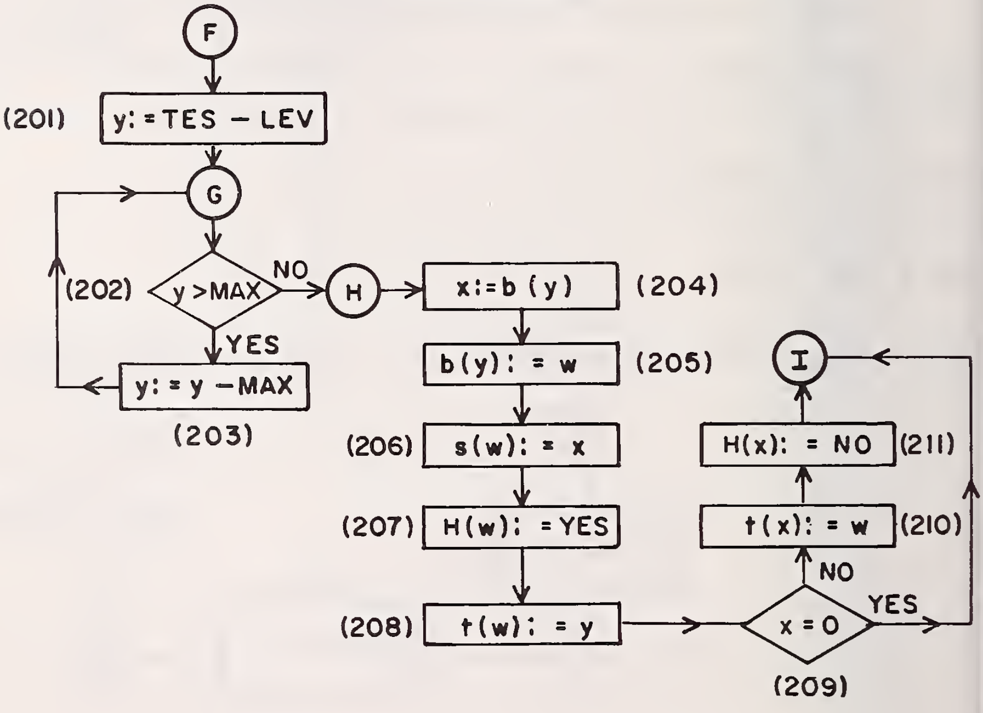
$b(z) = 0,\ 1 \leq z \leq MAX + 1$

$H(v) = NO$, for all v

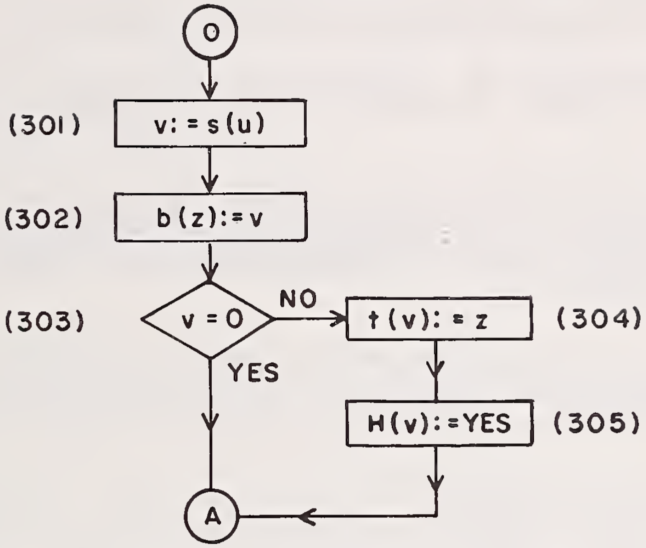
REMOVE w



ENTER w



DELETE u



This work has greatly profited from continual discussions with the staff of the Operations Research Section of the National Bureau of Standards. In addition, Dr. Bernard M. Levin of the National Bureau of Standards made his extensive bibliography available during the early stages of the work. Valuable insight was gained from experiments conducted and communicated by C.T. Zahn, Jr. The use of binary tree sorting was suggested by E. McCreight. The work was started under the support of the United States Post Office (the Office of Research and Engineering), and parts were also supported by the United States Department of Transportation (Northeast Corridor Transportation Project). Our thanks for excellent typing go to Mrs. Dolores E. Harrison.

10. REFERENCES

[1] Bellman, R., On a routing problem, Quart. Appl. Math. 16, 87-90 (1958).

[2] Bellman, R., and R. Kalaba., On k$\underline{\text{th}}$ best policies, J. Soc. Indust. Appl. Math. 8, 582-588 (1960).

[3] Bentley, D.L., and Kenneth L. Cooke, Convergence of successive approximations in the shortest route problem, J. Math. Anal. Appl. 10, 269-274 (1965).

[4] Berge, C., Theorie des graphes et ses applications, Dunod, Paris, (1958).
Translated, Theory of graphs and its applications, Methuen, London, (1962).

[5] Bock, F., H. Kantner, and J. Haynes, An algorithm (the r$\underline{\text{th}}$ best path algorithm) for finding and ranking paths through a network, (1957) Research report, ARMOUR RESEARCH FOUNDATION, Technology Center, Chicago, Ill.

[6] Clarke, S., A. Krikorian, and J. Rausen, Computing the N best loopless paths in a network, J. Soc. Indust. Appl. Math. 11, 1096-1102 (1963).

[7] Dantzig, G.B., On the shortest route through a network, Management Sci. 6, 187-190 (1960).

[8] ______, Discrete-variable extremum problems, Operations Res. 5, 268-273 (1957).

[9] ______, All shortest routes in a graph, Operations Res. House, Stanford University, Technical Report No. 66-3, November (1966).

[10] Dial, Robert B., Algorithm 360 Shortest path forest with topological ordering [H], Comm. ACM 12, 632-633 (1969).

[11] Dijkstra, E.W., A note on two problems in connexion with graphs, Numer. Math 1, 269-271 (1959).

[12] Dreyfus, Stuart E., An appraisal of some shortest-path algorithms, Operations Res. 17, 395-412 (1969).

[13] Farbey, B.A., A.H. Land, and J.D. Murchland, The Cascade Algorithm for finding all shortest distances in a directed graph, Management Science 14, 19-28 (1967).

[14] Floyd, R.W., "Algorithm 97, shortest path", Comm. ACM, 5, 345 (1962).

[15] Ford, L.R., Jr., and D.R. Fulkerson, Flows in networks, Princeton University Press, Princeton, (1962).

[16] Fulkerson, D.R., Flow networks and combinatorial operations research, Amer. Math. Monthly, 73, 115-138 (1966).

[17] Foulkes, J.D., W. Prager, and W.H. Warner, On bus schedules, Management Science 1, 41-48, (Oct. 1954).

[18] Goldman, A.J., and Christoph Witzgall, Most profitable routing before maintenance, abstracted in Bulletin of Operations Res. Soc. of Am., Operations Res., 13 supplement 1, B-82 (1965).

[19] Hardgrave, W.W., and G.L. Nemhauser, On the relation between the traveling salesman and the longest path problems, Operations Res. 10, 647-657 (1962).

[20] Hart, Peter E., Nils J. Nilsson, and Bertram Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions of Systems Science and Cybernetics SSC-4, 100-107 (1968).

[21] Hitchner, Lewis E., A comparative investigation of the computational efficiency of shortest path algorithms, Report ORC 68-25, Operations Research Center, University of California, Berkley (1968).

[22] Hoffman, W., and R. Pavley, Applications of digital computers to problems in the study of vehicular traffic, Proc. Western Joint Computer Conf., 159-161 (May 1958).

[23] _______, A method for the solution of the nth best path problem, J. Assoc. Comput. Mach. 6, 506-514 (1959).

[24 Hu, T.C., Revised matrix algorithms for shortest paths, SIAM J. Appl. Math. 15, 207-218 (1967).

[25] ________, A decomposition algorithm for shortest paths in a network, Operations Res. 16, 91-102 (1968).

[26] ________, Integer programming and network flows, Addison-Wesley, Reading, Mass. (1969).

[27] Hu, T.C. and W.T. Torres, Shortcut in the decomposition algorithm for shortest paths in a network, IBM J. of Res. Develop., 387-390 (1969).

[28] Johnson, E.L., Networks and basic solutions, Operations Res. 14, 619-623 (1966).

[29] Klee, V., A "string algorithm" for shortest path in directed networks, Operations Res. 12, 428-432 (1964).

[30] Knuth, Donald E., The art of computer programming, Vol.1, Fundamental algorithms, Addison-Wesley, Reading, Mass., 305-406 (1968).

[31] Land, A.H. and S.W. Stairs, The extension of the cascade algorithm to large graphs, Management Science 14, 29-33 (1967).

[32] Levin, B.M., and S. Hedetniemi, Determining fastest routes using fixed schedules, Spring Joint Computer Conference (1963).

[33] Loubal, Peter, A network evaluation procedure, Highway Research Record 205, 96-109 (1967).

[34] Luccio, F., On some iterative methods for the determination of optimal paths through a network, Calcolo 3, 31-48 (1966).

[35] Martin, B.V., Minimum path algorithms for transportation planning, MIT, Dept. of Civil Engineering, Highway Transportation Project (1963).

[36] McCarthy, J.,et. al., LISP 1.5 Programmer's Manual, MIT Press, Cambridge, Mass. (1962).

[37] Mickle, Marlin, Tsung-Wei Sze, and Donald E. Rathbone, Neighborhood Storage, IEEE Transactions of Systems Science and Cybernetics SSC-4, 138-144 (1968).

[38] Mills, G., A decomposition algorithm for the shortest route problem, Operations Res. 14, 279-291 (1966).

[39] Minty, G.J., A comment on the shortest-route problem, Operations Res. 5, 724 (1957).

[40] _____, A variant on the shortest-route problem, Operations Res. 6, 882 (1958).

[41] Moore, E.F., The shortest path through a maze, Proceedings of an international symposium on the theory of switching, Part II, April 2-5 (1957), The Annals of the Computation Laboratory of Harvard University 30, Harvard University Press (1959).

[42] Murchland, J.B., The shortest-route problem-An addendum, Operations Res. 9, 129-132 (1961).

[43] Nemhauser, George L., A generalized permanent label setting algorithm for the shortest path between specified nodes, J. of Math. Appl. 38, 328-334 (1972).

[44] Nicholson, T.A.J., Finding the shortest route between two points in a network, Computer J. 9, 275-280 (1966).

[45] Pandit, S.N.N., A new matrix calculus, J. Soc. Indust. Appl. Math. 9, 632-639 (1961).

[46] ______, Some observations on the routing problem, Operations Res. 10, 726 (1962).

[47] ______, Some observations on the longest path problem, Operations Res. 12, 361-364 (1964).

[48] Peart, R.M., P.H. Randolf, and T.E. Bartlett, The shortest-route problem, Operations Res. 8, 866-868 (1960).

[49] Pollack, M., The maximum capacity through a network, Operations Res. 8, 733-736 (1960).

[50] _______, The k^{th} best route through a network, Operations Res. 9, 578-580 (1961).

[51] _______, Solutions of the k^{th} best route through a network-a review, J. Math. Anal. Appl. 3, 547-599 (1961).

[52] Pollack, M., and W. Wiebenson., Solutions of the shortest-route problem-a review, Operations Res. 8, 224-230 (1960).

[53] ______________________________, Comments on "the shortest-route problem" by Peart, Randolph, and Bartlett, Operations Res. 9, 411-412 (1961).

[54] Sakarovitch, M., The k shortest routes and the k shortest chains in a graph, Operations Research Center, University of California, Berkley, Report ORC 66-32 (1966).

[55] Saksena, J.P. and Santosk Kumar, The routing problem with 'K' specified nodes, Operations Res. 14, 909-913 (1966).

[56] Scions, H.I., The compact representation of a rooted tree and the Transportation Problem, Internat. Symp. on Math. Programming, London, (1964).

[57] Shimbel, A., Structure in communication nets, Proceedings of the Symposium on Information Networks, N.Y., Brooklyn Polytechnic Institute (1954).

[58] Suurballe, J.W., Network algorithms, International Research Institute, Box 302, Far Hills, N.J.

[59] Traffic assignment manual for applications with a large, high-speed computer, U.S. Department of Commerce, Bureau of Public Roads, Office of Planning, Urban Planning Division (1964).

[60] Weimer, D.L., A serial technique to determine minimum paths, Comm. ACM 6, 663-665 (1963).

[61] Whiting, P.D., and J.A. Hillier, A method for finding the shortest route through a road network, Operational Research Quarterly, 11, 37-40 (1960).

[62] Yaged, B., Traffic flow information for minimum cost routing procedures, Automatica 5, 167-173 (1969).

APPENDIX

FORTRAN PROGRAMS FOR SEVEN LABELING SHORTEST PATH ALGORITHMS

This appendix contains listings of FORTRAN programs for the seven labeling algorithms upon which the experiments described in section 7 were performed. The programs have been included both to fully document the experimental runs and to provide programs for readers who need to use a shortest path algorithm. Copies of the FORTRAN card decks may be obtained from:

Judith Gilsinn
Operations Research Section, Applied
Mathematics Division
Room 428, Administration Building
National Bureau of Standards
Washington, D.C. 20234

Input and output routines have not been included in this appendix, since they are much more machine-and user-dependent than the algorithms. The programs listed here utilize BLANK COMMON to pass data to and from the algorithms. The arrays A, N, and L (corresponding to a, n, and ℓ in the text) together with the variables NODE, the number of nodes, LINK, the number of arcs, and MAX, one plus the maximum arc length, contain the network description. The algorithm then calculates the tree rooted at R (corresponding to r) and stores it in the arrays D and P (corresponding to d and p in the text).

```
      SUBROUTINE C1
      COMMON N(10000),L(10000),D(3900),P(3900),A(3900),R,
     1        NODE,LINK,MAX,T(4000)
      INTEGER    A,P,D,R,V,AUX,E,W,TES
      LOGICAL    T,MEM
C  INITIALIZATION
      DO 6 V=1,NODE
      P(V)=0
      D(V)=999999999
      T(V)=.FALSE.
6     CONTINUE
      D(R)=0
      T(R)=.TRUE.
C  START ALGORITHM
7     MEM=.FALSE.
      DO 10 V=1,NODE
      IF (.NOT.T(V)) GO TO 10
      AUX=D(V)
      IDOWN=A(V)
      IF (IDOWN.EQ.0) GO TO 9
      M=V+1
71     IUP=A(M)-1
      IF (IUP.GT.-1) GO TO 72
      M=M+1
      GO TO 71
72    DO 8 E=IDOWN,IUP
      W=N(E)
      TES=L(E)+AUX
      IF (TES.GE.D(W)) GO TO 8
      D(W)=TES
      P(W)=V
      T(W)=.TRUE.
      MEM=.TRUE.
8     CONTINUE
9     T(V)=.FALSE.
10    CONTINUE
      IF (MEM) GO TO 7
      RETURN
      END
      SUBROUTINE C2
      COMMON N(10000),L(10000),D(3900),P(3900),A(3900),R,
     1        NODE,LINK,MAX,T(4000),B(4000)
      INTEGER  A,P,D,R,V,U,AUX,E,W,TES,B,Z
      LOGICAL T
C   INITIALIZATION
      DO 6 V=1,NODE
      P(V)=0
      T(V)=.FALSE.
      D(V)=999999999
6     CONTINUE
      D(R)=0
C   START ALGORITHM
      V=1
      Z=V
      U=R
7     AUX=D(U)
      IDOWN=A(U)
      IF (IDOWN.EQ.0) GO TO 11
      M=U+1
8     IUP=A(M)-1
      IF (IUP.GT.-1) GO TO 9
      M=M+1
      GO TO 8
9     DO 10 E=IDOWN,IUP
      W=N(E)
      TES=L(E)+AUX
      IF (D(W).LE.TES) GO TO 10
      D(W)=TES
      P(W)=U
      IF (T(W)) GO TO 10
      T(W)=.TRUE.
      B(Z)=W
      Z=Z+1
      IF (Z.GT.NODE) Z=1
10     CONTINUE
11    IF (V.EQ.Z) RETURN
      U=B(V)
      T(U)=.FALSE.
      V=V+1
      IF (V.GT.NODE) V=1
      GO TO 7
      END
      SUBROUTINE C3
      COMMON N(10000),L(10000),D(3900),P(3900),A(3900),R,
     1        NODE,LINK,MAX,F(4000),G(4000),B(4000),H(4000)
      INTEGER  A,P,D,R,V,F,G,E,W,TES,AUX,X,DEL,T,S
      LOGICAL  B,H,MEM
      DO 1 V=1,NODE
      D(V)=999999999
      P(V)=0
      B(V)=.FALSE.
      H(V)=.FALSE.
 1    CONTINUE
      D(R)=0
      F(R)=R
      H(R)=.TRUE.
      V=R
 2    MEM=.FALSE.
 20   IF (.NOT.H(V)) GO TO 12
      AUX=D(V)
      IDOWN=A(V)
      IF (IDOWN.EQ.0) GO TO 11
      I=V+1
 3    IUP=A(I)-1
      IF (IUP.GT.-1) GO TO 4
      I=I+1
      GO TO 3
 4    DO 10 E=IDOWN,IUP
      W=N(E)
      TES=L(E)+AUX
      X=W
      IF (D(W).GE.999999999) GO TO 9
 5    DEL=D(W)-TES
      IF (DEL.LE.0) GO TO 10
C   SEVER
 6    B(X)=.TRUE.
      T=F(X)
      IF (T.EQ.R) GO TO 7
      I=P(T)
      IF (.NOT.B(I)) GO TO 7
      D(T)=D(T)-DEL
      H(T)=.TRUE.
      X=T
      GO TO 6
 7    S=G(W)
      F(S)=T
      G(T)=S
      T=W
 8    B(T)=.FALSE.
      IF (T.EQ.X) GO TO 9
      T=F(T)
      GO TO 8
```

```
C   ATTACH
9       S=F(V)
        F(X)=S
        G(S)=X
        F(V)=W
        G(W)=V
        P(W)=V
        D(W)=TES
        H(W)=.TRUE.
        MEM=.TRUE.
10      CONTINUE
11      H(V)=.FALSE.
12      V=G(V)
        IF (V.NE.R) GO TO 20
        IF (MEM) GO TO 2
        RETURN
        END
```

```
      SUBROUTINE S1
      COMMON N(10000),L(10000),D(3900),P(3900),A(3900),R,
     1        NODE,LINK,MAX,B(4000),T(4000)
      INTEGER     A,P,D,R,V,Z,S,U,X,Y,AUX,TEM,TES,E,W,B,T
C   INITIALIZATION
      DO 6 V=1,NODE
      P(V)=0
      D(V)=999999999
6     CONTINUE
       D(R)=0
C   START ALGORITHM
      Z=1
      B(Z)=R
      S=Z
      T(R)=Z
7     U=B(S)
      IDOWN=A(U)
      IF (IDOWN.EQ.0) GO TO 15
      AUX=D(U)
      I=U+1
8     IUP=A(I)-1
      IF (IUP.GT.-1) GO TO 9
      I=I+1
      GO TO 8
9     DO 14 E=IDOWN,IUP
      W=N(E)
      TES=L(E)+AUX
      TEM=D(W)
      IF (TEM.GE.999999999) GO TO 10
      IF (TEM.LE.TES) GO TO 14
      GO TO 11
10    Z=Z+1
      B(Z)=W
      T(W)=Z
11    D(W)=TES
      P(W)=U
C   SORT
      Y=T(W)
12    X=Y
      Y=Y-1
      V=B(Y)
      IF (D(V).LE.TES) GO TO 13
      B(X)=V
      T(V)=X
      GO TO 12
13    B(X)=W
      T(W)=X
14     CONTINUE
15    IF (S.EQ.Z) RETURN
      S=S+1
      GO TO 7
      END
      SUBROUTINE S2
      COMMON N(10000),L(10000),D(3900),P(3900),A(3900),R,
     1       NODE,LINK,MAX,B(4000),T(4000)
      INTEGER  E,P,TES,Z,A,H,Q,S,U,AUX,R,T,V,B,TEM,W,D,TEN,Y
C   INITIALIZATION
      DO 1 V=1,NODE
      P(V)=0
      IF (V.EQ.R) GO TO 1
      D(V)=99999999
1     CONTINUE
      D(R)=0
C   START ALGORITHM
      Z=1
      U=R
2     IDOWN=A(U)
      IF (IDOWN.EQ.0) GO TO 10
      AUX=D(U)
      I=U+1
3     IUP=A(I)-1
      IF (IUP.GT.-1) GO TO 4
      I=I+1
      GO TO 3
4     DO 9 E=IDOWN,IUP
      W=N(E)
      TES=L(E)+AUX
      TEM=D(W)
      IF (TEM.LT.999999999) GO TO 5
      S=Z
      Z=Z+1
      GO TO 6
5     IF (TEM.LE.TES) GO TO 9
      S=T(W)
6     D(W)=TES
      P(W)=U
C   SIFT UP
7     H=S/2
      IF (H.LE.0) GO TO 8
      V=B(H)
      TEM=D(V)
      IF (TEM.LE.TES) GO TO 8
      B(S)=V
      T(V)=S
      S=H
      GO TO 7
8     B(S)=W
      T(W)=S
9     CONTINUE
10    Z=Z-1
      IF (Z.EQ.0) RETURN
      Q=B(Z)
      U=B(1)
```

```
C  SIFT DOWN
      S=1
      TES=D(Q)
11    H=S*2
      IF (H.GE.7) GO TO 13
      V=B(H)
      TEM=D(V)
      K=H+1
      IF (K.GE.7) GO TO 12
      Y=B(K)
      TEN=D(Y)
      IF (TEN.GF.TEM) GO TO 12
      TEM=TEN
      V=Y
      H=K
12    IF (TES.LF.TEM) GO TO 13
      B(S)=V
      T(V)=S
      S=H
      GO TO 11
13    B(S)=Q
      T(Q)=S
      GO TO 2
      END
      SUBROUTINF S3
      COMMON N(10000).L(10000),D(3900),P(3900),A(3900),R,
     1         NODE,LINK,MAX,S(4000),T(4000),U(4000),H(4000)
      INTEGER  F,P,TET,Y,F,Q,S,U,Z,A,R,T,V,AUX,TEM,W,D.TES.X
      LOGICAL  H
      DO 1 V=1,NODE
      D(V)=999999999
      P(V)=0
      H(V)=.FALSE.
      S(V)=0
      T(V)=0
      U(V)=0
1     CONTINUE
      D(R)=0
      H(R)=.TRUF.
      F=0
      Z=0
      V=R
2     IDOWN=A(V)
      IF (IDOWN.EQ.0) GO TO 22
      AUX=D(V)
      I=V+1
3      IUP=A(I)-1
      IF (IUP.GT.-1) GO TO 4
      I=I+1
      GO TO 3
4     DO 21 E=IDOWN,IUP
      W=N(E)
      TES=L(E)+AUX
      TEM=D(W)
      IF (TEM.LT.999999999) GO TO 5
      Q=Z
      Y=0
      H(W)=.TRUF.
      GO TO 13
5     IF (TEM.LF.TES) GO TO 21
C   REMOVE W
      X=S(W)
      Q=T(W)
      IF (H(W)) GO TO 7
      S(Q)=X
      IF (X.NE.0) T(X)=Q
6     Y=Q
      Q=T(Y)
      IF (.NOT.H(Y)) GO TO 6
      H(W)=.TRUF.
      GO TO 13
```

```
7     Y=U(W)
      IF (X.EQ.0) GO TO 10
      H(X)=.TRUE.
      T(X)=Q
      U(X)=Y
      IF (Y.EQ.0) GO TO 8
      T(Y)=X
      GO TO 9
8     Z=X
9     IF (Q.NE.0) U(Q)=X
      Y=X
      GO TO 13
10    IF (Y.EQ.0) GO TO 11
      T(Y)=Q
      GO TO 12
11    Z=Q
12    IF (Q.NE.0) U(Q)=Y
C  ENTER W
13    IF (Q.EQ.0) GO TO 15
      TET=D(Q)
      IF (TET.LT.TES) GO TO 16
      IF (TET.EQ.TES) GO TO 14
      Y=Q
      Q=T(Y)
      GO TO 13
14    I=T(Q)
      IF (I.NE.0)   U(I)=W
24    IF (Y.EQ.0) GO TO 25
      T(Y)=W
      GO TO 26
25    Z=W
26    T(W)=I
      U(W)=Y
      S(W)=Q
      T(Q)=W
      H(Q)=.FALSE.
      GO TO 20
15    F=W
      GO TO 17
16    U(Q)=W
17    IF (Y.EQ.0) GO TO 18
      T(Y)=W
      GO TO 19
18    Z=W
19    T(W)=Q
      S(W)=0
      U(W)=Y
```

```
20      D(W)=TES
        P(W)=V
21      CONTINUE
22      IF (F.EQ.0) RETURN
        V=F
        IF (V.EQ.7) Z=0
        F=T(F)
        IF (F.NE.0) GO TO 2
        F=U(V)
        IF (F.EQ.0) GO TO 2
        T(F)=0
23      IF (S(F).FQ.0) GO TO 2
        F=S(F)
        GO TO 23
        END
```

```
      SUBROUTINF S4
      COMMON N(10000),L(10000),D(3900),P(3900),A(3900),R,
     1        NODE,LINK,MAX,S(4000),T(4000),B(4000),H(4000)
      INTEGER  F,TES,Y,A,U,Z,AUX,P,S,V,B,R,T,W,D,TEM,X
      LOGICAL  MEM,H
      DO 1 U=1,NODE
      D(U)=999999999
      H(U)=.FALSE.
      S(U)=0
      T(U)=0
      P(U)=0
1     CONTINUE
      D(R)=0
      DO 2 Z=1,MAX
      B(Z)=0
2     CONTINUE
C  START ALGORITHM
      MEM=.FALSF.
      LEV=0
      Z=1
      U=R
3     IDOWN=A(U)
      IF (IDOWN.EQ.0) GO TO 13
      AUX=D(U)
      I=U+1
4     IUP=A(I)-1
      IF (IUP.GT.-1) GO TO 5
      I=I+1
      GO TO 4
5     DO 12 E=IDOWN,IUP
      W=N(E)
      TES=L(E)+AUX
      TEM=D(W)
      IF (TEM.GF.999999999) GO TO 8
      IF (TEM.LF.TES) GO TO 12
C  REMOVE W
      X=S(W)
      Y=T(W)
      IF (H(W)) GO TO 6
      S(Y)=X
      IF (X.NE.0) GO TO 7
      GO TO 8
6     B(Y)=X
      IF (X.EQ.0) GO TO 8
      H(X)=.TRUF.
7     T(X)=Y
```

```
C  ENTER W
8      Y=TES-LEV
9      IF (Y.LE.MAX) GO TO 10
       Y=Y-MAX
       GO TO 9
10     X=B(Y)
       B(Y)=W
       S(W)=X
       H(W)=.TRUF.
       T(W)=Y
       IF (X.EQ.0) GO TO 11
       T(X)=W
       H(X)=.FALSE.
11     D(W)=TES
       P(W)=U
       MEM=.TRUE.
12     CONTINUE
13     U=B(Z)
       IF (U.NE.0) GO TO 14
       Z=Z+1
       IF (Z.LE.MAX) GO TO 13
       IF (.NOT.MEM) RETURN
       MEM=.FALSF.
       LEV=LEV+MAX
       Z=1
       GO TO 13
C  DELETE U
14     V=S(U)
       B(Z)=V
       IF (V.EQ.0) GO TO 3
       T(V)=Z
       H(V)=.TRUF.
       GO TO 3
       END
```

☆ U. S. GOVERNMENT PRINTING OFFICE : 1973--511-325/242

U.S. DEPT. OF COMM.
BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NO.
NBS TN-772

2. Gov't Accession No.

3. Recipient's Accession No.

4. TITLE AND SUBTITLE

A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees

5. Publication Date
May 1973

6. Performing Organization Code

7. AUTHOR(S)
Judith Gilsinn and Christoph Witzgall

8. Performing Organization

9. PERFORMING ORGANIZATION NAME AND ADDRESS

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

10. Project/Task/Work Unit No.
2050152

11. Contract/Grant No.

12. Sponsoring Organization Name and Address

Same as No.9

13. Type of Report & Period Covered
Final

14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

16. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)

Many applications in transportation and communication require the calculation of shortest routes between points in a network, and several algorithms for the solution of this problem exist in the literature. This paper examines one class of such algorithms, that which calculates a shortest route from one point in the network to all other intersection points. Computer data handling techniques which can be used to improve the two basic algorithms in this class are investigated. Results of computer timing runs on various types and sizes of networks are compared, and the differences, sometimes of an order of magnitude, are analyzed. Detailed flowcharts and computer programs of the tested algorithms are also included.

17. KEY WORDS (Alphabetical order, separated by semicolons)

Algorithms; networks; paths; shortest-paths; trees.

18. AVAILABILITY STATEMENT

☒ UNLIMITED.

□ FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NTIS.

19. SECURITY CLASS (THIS REPORT)
UNCLASSIFIED

20. SECURITY CLASS (THIS PAGE)
UNCLASSIFIED

21. NO. OF PAGES
92

22. Price
$1.25 Domestic postpaid
$1.00 G.P.O. Bookstore

NBS TECHNICAL PUBLICATIONS

PERIODICALS

JOURNAL OF RESEARCH reports National Bureau of Standards research and development in physics, mathematics, and chemistry. Comprehensive scientific papers give complete details of the work, including laboratory data, experimental procedures, and theoretical and mathematical analyses. Illustrated with photographs, drawings, and charts. Includes listings of other NBS papers as issued.

Published in two sections, available separately:

- **Physics and Chemistry (Section A)**

Papers of interest primarily to scientists working in these fields. This section covers a broad range of physical and chemical research, with major emphasis on standards of physical measurement, fundamental constants, and properties of matter. Issued six times a year. Annual subscription: Domestic, $17.00; Foreign, $21.25.

- **Mathematical Sciences (Section B)**

Studies and compilations designed mainly for the mathematician and theoretical physicist. Topics in mathematical statistics, theory of experiment design, numerical analysis, theoretical physics and chemistry, logical design and programming of computers and computer systems. Short numerical tables. Issued quarterly. Annual subscription: Domestic, $9.00; Foreign, $11.25.

TECHNICAL NEWS BULLETIN

The best single source of information concerning the Bureau's measurement, research, developmental, cooperative, and publication activities, this monthly publication is designed for the industry-oriented individual whose daily work involves intimate contact with science and technology—*for engineers, chemists, physicists, research managers, product-development managers, and company executives.* Includes listing of all NBS papers as issued. Annual subscription: Domestic, $6.50; Foreign, $8.25.

NONPERIODICALS

Applied Mathematics Series. Mathematical tables, manuals, and studies.

Building Science Series. Research results, test methods, and performance criteria of building materials, components, systems, and structures.

Handbooks. Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications. Proceedings of NBS conferences, bibliographies, annual reports, wall charts, pamphlets, etc.

Monographs. Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

National Standard Reference Data Series. NSRDS provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated.

Product Standards. Provide requirements for sizes, types, quality, and methods for testing various industrial products. These standards are developed cooperatively with interested Government and industry groups and provide the basis for common understanding of product characteristics for both buyers and sellers. Their use is voluntary.

Technical Notes. This series consists of communications and reports (covering both other-agency and NBS-sponsored work) of limited or transitory interest.

Federal Information Processing Standards Publications. This series is the official publication within the Federal Government for information on standards adopted and promulgated under the Public Law 89–306, and Bureau of the Budget Circular A–86 entitled, Standardization of Data Elements and Codes in Data Systems.

Consumer Information Series. Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

BIBLIOGRAPHIC SUBSCRIPTION SERVICES

The following current-awareness and literature-survey bibliographies are issued periodically by the Bureau:

Cryogenic Data Center Current Awareness Service (Publications and Reports of Interest in Cryogenics). A literature survey issued weekly. Annual subscription: Domestic, $20.00; foreign, $25.00.

Liquefied Natural Gas. A literature survey issued quarterly. Annual subscription: $20.00.

Superconducting Devices and Materials. A literature survey issued quarterly. Annual subscription: $20.00. Send subscription orders and remittances for the preceding bibliographic services to the U.S. Department of Commerce, National Technical Information Service, Springfield, Va. 22151.

Electromagnetic Metrology Current Awareness Service (Abstracts of Selected Articles on Measurement Techniques and Standards of Electromagnetic Quantities from D-C to Millimeter-Wave Frequencies). Issued monthly. Annual subscription: $100.00 (Special rates for multi-subscriptions). Send subscription order and remittance to the Electromagnetic Metrology Information Center, Electromagnetics Division, National Bureau of Standards, Boulder, Colo. 80302.

Order NBS publications (except Bibliographic Subscription Services) from: Superintendent of Documents, Government Printing Office, Washington, D.C. 20402.