# A Comprehensive Program for Network Problems

*By* E. W. Solomon

**This article describes in general terms, a program which has been written for the Pegasus digital computer, which will solve several of the more common problems associated with networks having weighted branches. All operations are carried out within the incidence matrix stored serially by rows in the computer.**

**After a brief introduction to the concepts and terminology of graph theory, a comparison is made between the two principal ways in which a network may be represented within a computer. The discussion on the form of output after the program has transformed the incidence matrix includes a description of a tree symbol more suited to construction in a computer than those given by Prüfer and by Loberman and Weinberger.**



## Introduction

In this paper we shall be interested in the problems associated with graphs and networks having weighted branches. Examples of practical problems falling into this category are electrical wiring problems involving minimization of materials, Knight's tour problems with minimum travelling and geographical shortest route finding. Several applications of a program of the type to be described are discussed at the end of the paper. Although close to experience, the concepts and especially the terminology appear to be relatively unfamiliar still. We therefore feel it necessary to follow the accepted pattern and to include a preface dealing with the elementary ideas of the field. Readers who have encountered previous articles on this topic can safely omit the following two sections entitled respectively Graphs and Networks.

The program to be described is designed to solve the following problems for a given network.

(1) To find the minimum route between two given points, subject to specified cuts in the network.
(2) To construct the minimum tree on a given node.
(3) To construct the minimum spanning sub-tree of the network.

The program is constructed so that additional problems can be solved using suitable subroutines which, however, have not so far been tested. These include an approximate solution to the Travelling Salesman problem and the problem of finding the $N$th minimal route between two points. (The Travelling Salesman problem requires the construction of an algorithm for discovering the minimal circuit which includes every node of the network.)

## Graphs

A *finite graph* is defined to be a set of $N_0$ objects, called *nodes*, between some or all pairs of which a certain symmetric relation exists. Two such nodes are said to be joined by a *branch*. It is usual to assign integers serially to the nodes, so that one can speak of a pair of nodes, 5 and 11 say, and of a branch (5, 11) if one exists. Nodes are drawn as points and branches as lines between them. Fig. 1 is an example of a graph. Furthermore, it is a "complete" graph because every node shares a branch with every other.

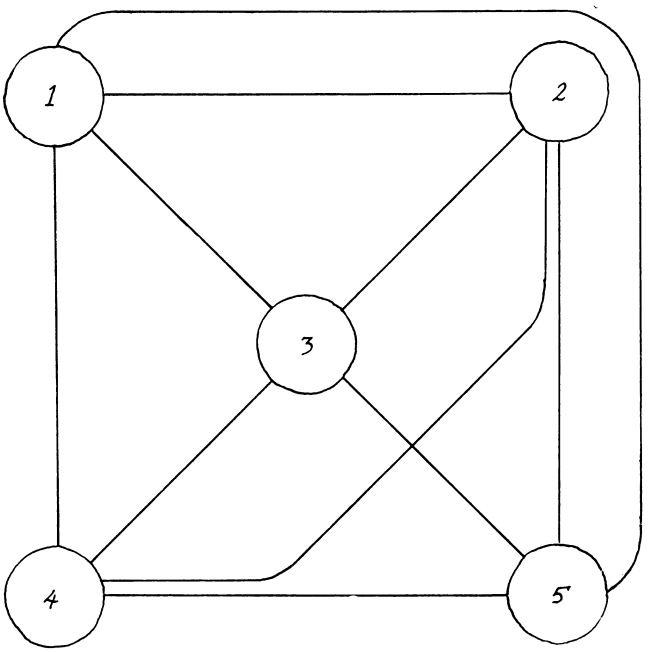


**Fig. 1.—The complete 5-graph**

The terminology introduced here is not completely universal, but appears to be the most widely used among applied mathematicians. Writers dealing with the subject from a pure mathematical point of view very often use the term *vertex* for a node, and branches are referred to as *edges*, *links*, or *arcs*. If one can get from a node $i$, to a node $j$, by passing along an ordered sequence of "$s$" branches, there exists a *path* between the nodes $i$ and $j$. With $i$ and $j$ distinct, the path is an *s-arc*. If the node $i$ is identical with the node $j$, the path is an *s-circuit*. A special case of an $s$-circuit is the 1-*circuit*. This is known as a *sling*, and would be represented by a branch which curves back on itself, so that both extremes lie on the same node. It will be assumed that, in all the graphs

and networks which are discussed here, there are no slings.

If there exists a path between every pair of nodes in a graph, the graph is said to be *connected* and it has *separability* $S = 1$. In general, a graph is divided into $S$ *separate pieces* between every pair of which no path exists. If also, the number of nodes is $N$, and the number of branches is $B$, one may define two useful quantities associated with a graph in the following way:

The *rank* of a graph is $N - S$.

The *nullity* of a graph is $B - (\text{Rank}) = B - N + S$.

An important concept is that of the *forest*. This is defined to be a graph of zero nullity, or in other words it is a graph with no circuit. A forest of separability one, is a *tree*. This, therefore, gives the picturesque result that a tree is always a forest but that a forest need not necessarily be a tree.

Since the present treatment concerns only connected graphs and networks, the term forest will not again be used. A tree can be redefined to be a graph in which there is one and only one path between every pair of nodes. This automatically precludes the existence of circuits. For a more thorough introduction to graphs, see Whitney (1932) and for a complete treatment of the topic, König (1950).

## Networks

Suppose that the relation between the nodes sharing a branch is not symmetric. The branches are now drawn as directed lines and the resulting configuration is called a *network*. A path in a network is defined as for a graph, but there is the obvious added requirement that the branches of the path must be directed in the same sense, so that one can no longer speak loosely of a path ($i$ to $j$), but must emphasize that the path is "from" $i$ "to" $j$, or vice versa. Care is also required in the recognition of a tree. In speaking of a tree "on" a node $k$, it is to be understood that every branch is directed away from $k$.

Fig. 2 is an example of a network with weighted branches.

The weights assigned to the branches are termed branch values. In the physical sense, such a value may be a distance, a transit time, a flow capacity, or some other numerical quantity. Note that in general the value of a branch $(i, j)$ need not be identical with the value of the branch $(j, i)$. Transit time along a crowded street depending on the general direction of flow of traffic, affords an obvious example of such assymmetry. Networks with weighted branches are not the only kind of network in which mathematicians are interested. A great deal of work has been carried out upon networks with weighted nodes, where the nodal values may be such varied entities as colours, storage capacities and different values of time variables including frequency.

One further term remains to be introduced. By the *valence* of a node $i$ is meant the number of branches on $i$ directed "away" from $i$, or alternatively the number of branches "emanating from" $i$.

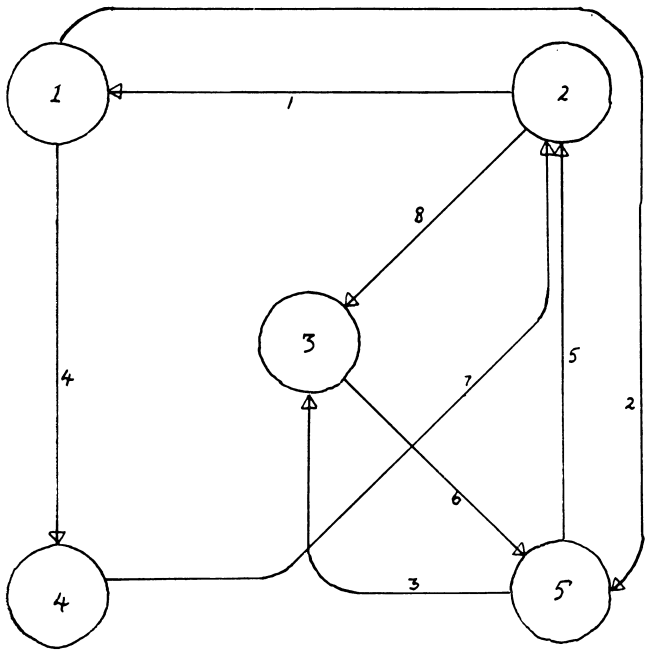


**Fig. 2.—An example of a network.**

## The Incidence Diagram and Matrix

*The incidence diagram* is the term given here to the array of binary symbols, zeros and non-zeros, for instance, distributed so as to represent the given graph or network. The 0's and 1's, say, are arranged in rows and columns so that a 1 at the intersection of the $i$th row and $j$th column indicates that there is a branch directed from $i$ to $j$. A zero denotes the absence of such a branch. The incidence diagram for the network of Fig. 2 is follows:

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 |

*The incidence matrix* is the square array of the branch values $d_{ij}$. Thus at the intersection of the $i$th row and the $j$th column is found a numerical value equal to the value of the branch $(i,j)$. The incidence matrix of the network of Fig. 2 is as follows:

| | | | | |
|---|---|---|---|---|
| $A$ | $A$ | $A$ | 4 | 2 |
| 1 | $A$ | 8 | $A$ | $A$ |
| $A$ | $A$ | $A$ | $A$ | 6 |
| $A$ | 7 | $A$ | $A$ | $A$ |
| $A$ | 5 | 3 | $A$ | $A$ |

$A = \infty$ if the branch values are distances or transit times.

$A = 0$ if the branch values are capacities.

Disregarding for the present purposes the frowns of the pure mathematicians, a graph is taken to be a special case of a network. A graph is a network for which the incidence matrix ($D_{ij}$) is symmetrical. A branch, therefore, between two nodes is considered directed in both senses, although it is drawn as undirected.

Algebraically, the condition that there are no slings, is that the diagonal of the incidence diagram is zero. Consequently, in storing an incidence diagram or matrix in the main store of a computer, better use can be made of the available storage space by omitting the diagonal. However, the increase in running time resulting from the more complex setting of the search modifier does not seem to justify the saving of space by this expedient.

This introduction to the algebraic representation is concluded with a brief reference to connectedness and sub-networks. Any network may contain a set of sub-networks, some of which can have the property that they are sub-graphs. That is, they can be drawn with undirected branches and the result is a mixed network. The sub-graph of a network is defined to be the sub-set of all branches $(i, j)$ and nodes $i, j$ lying on them with equal values $d_{ij} = d_{ji}$.

It is a slow process to calculate from an inspection of the incidence matrix, the separability of the sub-graph of a network, because we must first define the sub-graph. The separability of a given network, on the other hand, is found fairly rapidly by direct inspection of the matrix, or by re-ordering the node numbers so that the incidence matrix is transformed to a partitioned square matrix with null off-diagonal blocks.

## The Representation of a Network within the Computer

There are two distinct ways in which details of a network may be retained in the computer memory. Clearly we must specify:

(1) the total number of nodes in the system,
(2) the pairs of nodes between which branches exist,
(3) the values of these branches.

The first method described is essentially the incidence matrix stored serially by rows, whereas in the second the order in which the information is stored is immaterial.

### *The Incidence Matrix Method*

Assuming for convenience that the matrix is stored from address 0, the address of a branch value $d_{ij}$ is

$$(N_0 + 1)(i - 1) + j.$$

This expression holds because nodes are numbered from 1, not zero, and each row of the incidence matrix is preceded by a word called the *valence word*, which gives the valence of the node corresponding to the row which follows. Clearly one must indicate in some way the absence of a branch $(j, k)$. This is most easily achieved in a computer by setting zero at the position $(N_0 + 1)(j-1) + k$, although in some applications the numerically correct quantity would be arbitrarily large.

Under this system, a network of $N_0$ nodes will require $N_0(N_0 + 1)$ words of storage space, irrespective of the number of branches in the network. In any operation upon the matrix where a non-zero element vanishes, the valence word of the corresponding row must, accordingly, be corrected. Suppose that there exists a modifier whose value is $(N_0 + 1)(i - 1) + j$, but that it is required to extract and examine the branch value of the branch $(j, i)$. The modifier is changed to $(N + 1)(j - 1) + i$ and this operation is called "transposing" the modifier. The significance of this action will become evident in the section "Operations on the Incidence Matrix."

### *The Branch-Word Method*

The second basic method necessitates the use of partitioned words (at any rate this is unavoidable in a computer of small or moderate size). A *branch word* is a word holding three quantities as follows:

$$(i, j, d_{ij}).$$

Clearly the amount of space within a word allocated to each quantity depends on the number of nodes in the network, and on the maximum branch values $d_{ij}$ occurring in it. In the present paper, all branch values are regarded as integers. In practice this never causes inconvenience because should a network with fractional $d_{ij}$ arise, it is always possible to scale up and round to the nearest integer when rounding errors can be regarded as negligible. The address of a branch word plays no part in defining the properties of the network since all the information is contained within the words themselves. This is not to say, however, that some form of ordering of the branch words would not be advantageous from the operational point of view.

Clearly, with the branch-word method of storage, the restriction on capacity is not on the number of nodes in a network as with the matrix method, but on the total number of branches contained in it. In order to compare the merits and disadvantages of the two schemes, let us suppose that there are 1,000 words available in which to store the particulars of a network.

Under the matrix method, it is possible to store any network of $N_0$ nodes where

$$N_0(N_0 + 1) = 1{,}000$$
$$N_0 = 31 \text{ to the lowest integer.}$$

Any network whatsoever of 31 nodes or less, including the complete 31-network, may be accommodated in this way. Furthermore, the entire word is available to store the branch value, and no unpacking of the word is needed. In any fundamental operation, such as the extraction of a branch value $d_{ij}$ and subsequent arithmetical treatment of it, followed by a transfer of the new value back to the main store, the bulk of the time is taken up by the transfer operations. It is easy to see, therefore, that the matrix method has a great advantage in speed, for whereas in the branch-word method a search of some kind is required involving transfers at each stage, the matrix method requires only that a modifier be set for at most two transfers.

There is one type of network for which the matrix method is very wasteful of storage space. This is the network obtainable from an ordinary two dimensional street map, if every junction is regarded as a node and every road between two such junctions as a branch. In such a network, it is very unlikely that any valence would exceed eight, say. For such a case, the best system of storage is a combination of the two basic methods so far described, provided, that is, that there are more nodes than can be accommodated in an incidence matrix. Suppose that the maximum valence is $V_{max}$. Assign $V_{max} + 1$ consecutive words to each node, where the words have the form

$$(j, d_{ij}).$$

The first word, containing the valence of the node $i$, is used to set a counter prior to the search for a particular branch $(i, k)$ say. The number of transfers in any search never exceeds $V_{max} + 2$, and the number of nodes in a maximal network, assuming 1,000 words of store, is

$$N_0 = \frac{1{,}000}{V_{max} + 1}$$
$$= 125$$

for a network where no node has more than seven outgoing branches.

A few networks are characterized by having $V_{max}$ considerably greater than the mean valency. A system of railways is a typical example of such a network. A tested method for representing this kind of network allocates $V_i$ words to node $i$ so that there is no wastage of space at all. A directory of $N_0$ words is needed, the $i$th word of which has the form,

$$(V_i, A_i)$$

where $V_i$ is the valence of node $i$, and $A_i$ is the address of the first branch word for node $i$. The branch words for node $i$ have the form.

$$(j, d_{ij}).$$

These are followed without a space by the branch words for node $i + 1$ which start in address $A_{i+1}$ as given in the directory.

The present program is designed to employ the matrix method of storage. For a description of a program which uses a form of branch word storage, read Hoffman and Pavley (1959).

### Two Useful Operations on the Incidence Matrix

The two operations on the incidence matrix which are the nucleus of the computer program, are now described. The first of these is based on the well-known Moore's Algorithm (Moore, 1957) for finding the shortest route through a maze, and the second employs an algorithm described, without proof, by Dijkstra (1959), for constructing the shortest spanning sub-tree of a graph.

In both operations, it is considered that there is a starting-point node, $i$ for instance. At the beginning of both operations, the address of the valence word of the $i$th row is the only non-zero element of a set $S$ called the *search set*. The search set is the set of valence word addresses of the rows of the incidence matrix upon which the algorithm is operating. The positive non-zero elements of the rows in $S$, excluding the valence word, are referred to as the elements $d_{Sj}$.

From here onwards, the two algorithms are described separately.

*Algorithm* 1

The following two operations are performed successively until one of two termination criteria is satisfied. These criteria are described after the algorithm.

(1) Using the addresses, stored as modifiers, in the set $S$, find the smallest element $d_{Sj}$, say $d_{Sj}(\min)$, ($d_{Sj}(min)$ need not be unique), and add this quantity to a cumulative register $D$, say.

(2) Replace all $d_{Sj}$ by

$$d'_{Sj} = d_{Sj} - d_{Sj}(\min):$$

let $n_r$ elements $d_{rj_1}, d_{rj_2}, \ldots d_{rj_{n_r}}$ vanish in row $r$, $r \epsilon S$. Carry out the following operations:

($a$) Replace $V_r$ by $V'_r = V_r - n_r$, all $r \epsilon S$.

($b$) Set the vanished elements equal to $-1$. (The negative quantity acts as a marker.)

($c$) Set to zero all elements $d_{ij} > 0$ for all $i$, and $j \epsilon J$ where $J$ is the set of distinct columns containing a negative marker ($-1$), introduced in the current cycle of the algorithm. Accordingly, correct the valance words for all rows $i$.

($d$) Remove from $S$ all rows for which the valence word has become zero and introduce into $S$ all *rows* $j \epsilon J$. (Introduce, that is, the valence word addresses of rows $j$, stored as modifiers.)

Expressed in the above form, the algorithm appears to be lengthy, but this is misleading. In fact, the modifiers are so set at any stage that the next operation can be carried out, after a simple addition or subtraction of a previously stored quantity, except in the operation (2$d$) which involves replacement in the set $S$.

The column zero procedure of (2$c$) is carried out by a separate subroutine.

The termination criteria are as follows:

Either (1) one of the $j, (j \epsilon J)$, is equal to a pre-set destination node $D$, say,

or (2) a branch counter, set originally at $N_0 - 1$, is exhausted.

If termination is as a result of (1), the contents of register $D$ equals the minimum sum of branch values of a route from the starting-point node to the destination node. From the distribution of the ($-1$)'s in the resultant matrix, it is possible to assess the particular minimum route. On the other hand, if termination occurred because the branch counter was exhausted, $D$ is equal to the sum of branch values of the minimum tree on the starting point node, and the distribution of ($-1$)'s defines the tree. The tree can be defined more

economically by means of a tree symbol consisting of $N_0 - 2$ node numbers which can be quickly obtained from the resultant matrix in a way described under the section "The Tree Symbol."

*Algorithm* 2

This algorithm is given by Dijkstra (1959), and is used for constructing the sub-tree connecting every node of a graph, such that the sum of the branch values is a minimum. This is the *minimum spanning sub-tree*. The method of Dijkstra appears to be the best so far published in respect of working space and speed. Other methods are described by Loberman and Weinberger (1957).

No mathematical justification for the algorithm is given in Dijkstra's paper, which is brief, and therefore a proof of the theorem upon which it is based, is included here.

*Theorem* 1

For every node $i$ of a connected graph $G$, the minimum branch on $i$ is a member of the minimum spanning sub-tree.

*Proof*

Given that the minimum spanning sub-tree of $G$ is $T$, we assume that there is a node in $G$ whose minimal branch is not a member of $T$. We show that this leads to a contradiction.

Consider any node $i$ with a valency (in $T$) of $V_i$. $T$ can be regarded as the sum of $V_i$ sub-trees $A$, $B$, $C$, . . . etc., the node $i$ and its branches $(i\alpha_r)$, $(i\beta_r)$, $(i\gamma_r)$, . . . etc., where $\alpha_r \epsilon A$, $\beta_r \epsilon B$, $\gamma_r \epsilon C$, and so on.

Any pair of the sub-trees $A$, $B$, $C$, . . . is connected in $T$ only via the node $i$.

Now suppose that the minimum branch on $i$ is $(i\alpha_s) \epsilon G$, but that $(i\alpha_s) \notin T$. (There is no loss of generality in taking the minimum branch on $i$ to end in sub-tree $A$.)

We therefore have $d_{i\alpha_s} < d_{i\alpha_r}$.

Denote by $A'$ the connected tree made up of $A$, the node $i$ and its branch in $T$, $(i\alpha_r)$. The nullity of $A'$ is zero.

Now let us construct the *graph* $A''$ by removing from $A'$ the branch $(i\alpha_r)$ and introducing the branch $(i\alpha_s)$. If $N(H)$ and $B(H)$ are the number of nodes and branches in a graph $H$, we have

$$N(A'') = N(A') \text{ and } B(A'') = B(A').$$

Furthermore, the separability is unaltered because by definition of a tree, every node in $A'$ is connected, and in transforming to $A''$ no node has been isolated, and consequently no circuit introduced.

The nullity of $A''$ is therefore the same as the nullity of $A'$, that is, zero. $A''$ is therefore a tree. $A''$ also has the property that the sum of its branch values is less than the corresponding sum for $A'$, consequently $T$ could not have been the minimal spanning tree of $G$. The result that the minimal spanning sub-tree contains the minimum branch on every node is thus proven, since there was no restriction on the choice of $i$.

The algorithm is now described in terms of operations on the incidence matrix. Dijkstra describes it without reference to the incidence matrix, but the following contains no essentially new material.

*The Algorithm*

Again there is a search set $S$ and let the *non-zero, positive branch* values contained in the rows tagged by the elements of $S$, be $d_{Sj}$.

The first operation lying outside the loop of the algorithm is to choose an arbitrary node and place the valence word address of its corresponding row in set $S$. At any stage of the algorithm, the latest addition to the set $S$ is denoted by $S_e$ and its complement in $S$ consisting of all rows previously tagged, is written ${}_cS_e$. No rows are removed from $S$ at any time.

Initially, the branch counter is set as $N_0 - 1$, since the result is a connected tree. A cumulative branch value word is set to zero and then the following two sets of operations are performed cyclically.

(1) Find the minimum $d_{Sj} = d_{i,j_m}$ say, where $i \epsilon S$ $j \epsilon (1, 2 \ldots N_0)$.
  (*a*) Add $d_{i,j_m}$ to the cumulative branch value word
  (*b*) Set $d_{i,j_m}$ and $d_{j_m,i} = -1$ (the $-1$ acts as a marker).
  (*c*) Reduce the branch counter by unity and exit from the algorithm if it becomes zero.
  (*d*) Include row $j_m$ in $S$. Thus $j_m = S_e$.

(2) Compare the magnitude of the branch value $d_{Se,j}$ with the values $d_{(cSe),j}$ for each $j \epsilon (1, 2, \ldots N_0)$. (Note: only one of the elements $d_{(cSe),j}$ is non-zero for any $j$; this is the only element considered for the comparison.)

(*a*) If for any $j$, $d_{Se,j} \leqslant d_{(cSe),j}$, set to zero the latter and its transpose element $d_{j,(cSe)}$.

(*b*) If for any $j$, $d_{Se,j} > d_{(cSe),j}$, set to zero the former and its transpose element, $d_{j,Se}$. Return to (1).

Throughout the algorithm the valence word need not be changed since it plays no part. On exit from the loop, however, all the valence words are corrected to equal the number of $-1$'s in their corresponding rows. This is a faster process than would result if the valence words were corrected each time an element vanished. Subsequently, the pattern of $-1$'s is transformed in the way described in the tree symbol section.

**The Program**

The program has four principal parts which are quite straightforward:

A. Input
B. Operation
C. Output
D. Re-entry

A, C and D are described in detail here. Section B will be easily understood from the section on operations

on the incidence matrix and by a study of the flow diagram of Fig. 3. Where possible, the material is presented in diagrammatic and tabular form.

A. *Input*

Assuming that the network has already been stored as an incidence matrix in the machine store, section A is "input" in the following sense.

The course of the program and the form of the final output is obviously to be governed by the nature of the particular problem which it is desired to solve. This information is given to the program in the form of two integers, corresponding to a starting-point node and a destination node, which are read during section A. Table 1 explains the way in which the program distinguishes the problem it is required to solve. In the table, $i$ and $j$ are two nodes so that $i$ and $j$ lie in the range $+1$ to $+N_0$. An unspecified destination $j = 0$ means effectively that termination criterion (1) of Algorithm (1) cannot be used to halt the process and so the result will be a minimum tree on $i$. Similar reasoning holds for the other cases. The Travelling Salesman problem is included although a suitable subroutine has not yet been tested. In fact, no exact algorithm for this problem exists other than testing every possible circuit through all $N_0$ nodes and it may be that one is not possible, although several ways of approximating to the desired solution have been described.

The input section also prepares various modifiers and counters from the quantity $N_0$ which accompanies the incidence matrix.

C. *Output*

There are two forms of output (*a*) and (*b*). (*a*) is used if the result is a single route, and (*b*) if the result is a tree. Output (*a*) is very simple. The first quantity printed is the sum of the branch values of the route; this is preceded by a "+" sign. The succession of nodes commencing with the destination and terminating with the starting-point is then printed without signs. The following is an example of the output for the shortest route from node 3 to node 1 for the network of Fig. 2.

$$+12$$
$$1$$
$$2$$
$$5$$
$$3$$

Clearly, in practice, one would not employ a computer for such a trivial case.

The form of output (*b*) requires a more detailed explanation.

*Output* (*b*). *The tree symbol*

Prüfer (1918) and Weinberger and Loberman (1957) have given tree symbols consisting of sequences of

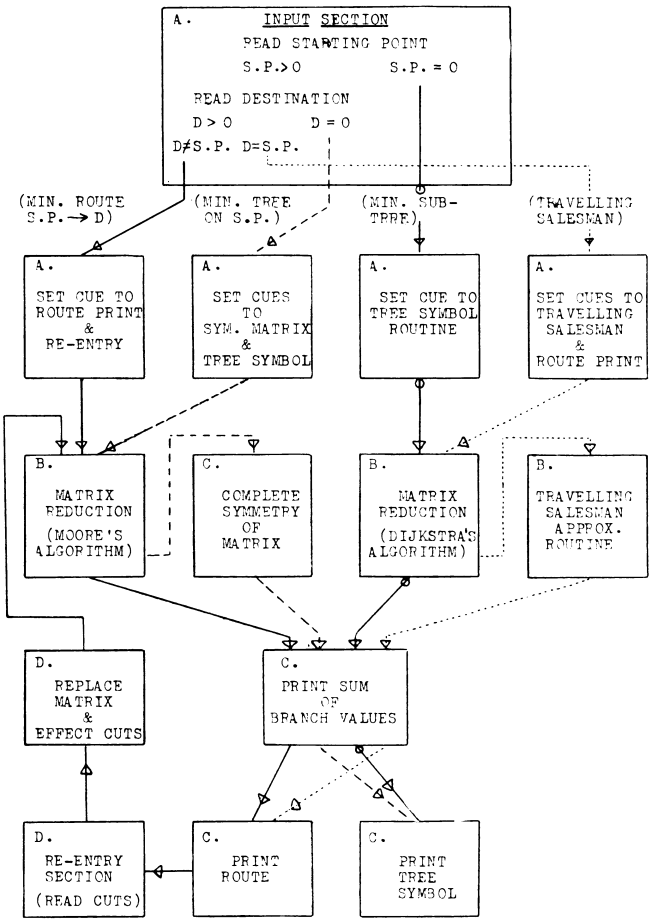


**Fig. 3.—Block diagram of program:**

**A. Input**
**B. Operation**
**C. Output**
**D. Re-entry**

$N_0 - 2$ node numbers. Each is unique and satisfactory for manual computation, but a shorter procedure as measured in computer time is possible. Note that here we are referring to graphical trees such as would be obtained for a "network" of electrical connections. The valence of each node is thus just the number of branches incident upon it and the incidence diagram is symmetrical. The method of obtaining the symbol corresponding to any tree, $T_0$ say, is to repeat the following operation until the symbol obtained contains $N_0 - 2$ node numbers.

Find the lowest numbered univalent node lying in the range $R$ where $R$ is defined in cycle $n$ to be the set of integers $r$ say, where $i_n < r \leqslant N_0$ and where $i_0 = 0$. Write down the number of the multivalent node to which it is connected. Set $i_{n+1} = i_n + 1$ in $R$. If there is no univalent node in $R$ before the complete symbol

**Table 1**

| STARTING-POINT | DESTINATION | THE PROBLEM | ACTION OF THE INPUT SECTION |
|---|---|---|---|
| $+i$ | $+j$ | Find the minimal route from $i$ to $j$. | Set cues to the subroutines:<br>(1) Route print.<br>If alternative routes are to be tested, set cues to re-entry sections D.<br>Enter matrix reduction Algorithm (1). |
| $+i$ | $-0$ | Construct the minimal tree on node $i$. | Set cues to the subroutines: (1) Make the matrix symmetrical with respect to negative elements.<br>(2) Print tree symbol.<br>Enter matrix reduction Algorithm (1) |
| $+0$ | — | Construct the minimum spanning sub-tree. | Set cues to the subroutines:<br>(1) Print tree symbol.<br>Enter matrix reduction Algorithm (2) |
| $+i$ | $+i$ | Approximate to the minimum Hamiltonian circuit. | Set cues to the subroutines:<br>(1) Travelling Salesman approximation.<br>(2) Route print.<br>Enter matrix reduction Algorithm (2). |

($N_0 - 2$ node numbers) has been obtained, reset $i_n = 0$ in $R$. The symbol is complete when $i_n$ is $i_{N_0-2}$.

Prüfer's method is essentially the same, but his range $R$ is invariant as (1 to $N$), consequently more time is spent in searching for univalent nodes since the search modifier is set back to zero at every stage. The methods of Weinberger and Loberman have similar disadvantages when programmed for a computer.

An example of the construction of a tree symbol is given for the tree of Fig. 4 in Table 2. The operations on the incidence matrix are described in parallel.

The tree corresponding to a given symbol can easily be constructed in the following way.

Search for the lowest node number not present in the symbol, but present in the complete list of nodes (1 to $N_0$).

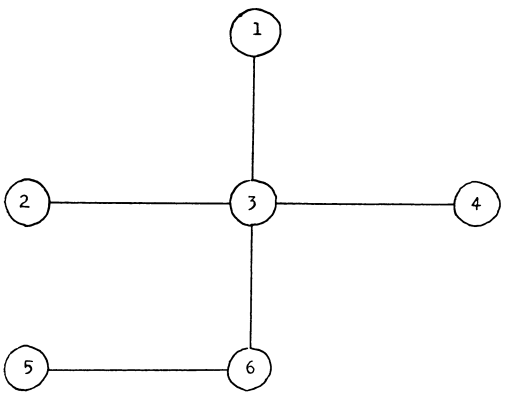


**Fig. 4.—A 6-node connected tree**

**Table 2**

**The Symbol for the Tree of Fig. 4**

| STEP | R | LOWEST UNIVALENT NODE IN R | CONNECTS TO NODE | SYMBOL | OPERATIONS IN MATRIX |
|---|---|---|---|---|---|
| 1 | (1, 6) | 1 | 3 | (3 ) | Subtract 1 from $V_1$ and $V_3$. Zero $d_{13}$ and $d_{31}$. |
| 2 | (2, 6) | 2 | 3 | (33 ) | Subtract 1 from $V_2$ and $V_3$. Zero $d_{23}$ and $d_{32}$. |
| 3 | (3, 6) | 4 | 3 | (333 ) | Subtract 1 from $V_4$ and $V_3$. Zero $d_{43}$ and $d_{34}$. |
| 4 | (4, 6) | 5 | 6 | (3336) | Symbol complete. *Exit from "Tree symbol routine."* |

This node connects to the first node of the symbol. Draw this and cross out the first mentioned node from the list of nodes and also the first node of the symbol. Now search for the next highest node not present in the symbol but present in the list of nodes. Draw the connection between this node and the second node of the symbol and perform the necessary crossing out. This process is continued until there are only two nodes uncrossed in the list; these two are connected. The result is the tree of the symbol. It is easily shown that the tree is unique by considering a general permutation of elements within the symbol and showing that all but the identity permutation yield a different tree. That the symbol is unique for a given tree is clear from the definition of the way in which the symbol is obtained.

D. *Re-entry*

*Non-minimal paths between two nodes*

Hoffman and Pavley have described an algorithm for finding not only the first but the $n$th minimal path between two nodes of a network. Their method depends upon the superposition on the network of the minimal tree on the starting node. It appears from their paper that they use the branch-word method of storing the network, but it would not be difficult to program their algorithm for a system using the incidence matrix method. It would also be feasible to use simply the "Moore's Algorithm" technique for solving this problem for small $n$, but the storage requirements increase in proportion to $n$. Another way of obtaining the $n$th minimal path would be to make permutations of trial cuts in the minimal and subsequently obtained paths. This would require no more storage space than the incidence matrix itself occupies, but the process would be very slow, especially for large $n$. A more realistic problem in terms of practical application, however, is the problem of finding the minimal route between two nodes, subject to cuts in the absolutely minimal route. For this purpose, the re-entry section of the program has been written. If this facility is to be used, the maximum size of the network is slightly reduced because the incidence matrix is duplicated in the machine store for quick replacement. Thus if, as in the case of Pegasus, there are say 6,400 words available for storing details of the network, a network of 79 nodes can be accommodated without duplication. With duplication, the maximal network is reduced to 56 nodes.

At the end of the route-printing routine, a stop is encountered. If it is required to make cuts, these are read in from tape in the form of pairs of integers $+i$, $+j$, etc., and the re-entry section of the program zeros the branches $d_{ij}$, etc., in the restored incidence matrix and corrects the valence words. The matrix reduction algorithm (1) is then re-entered. The process can be continued indefinitely.

## General Remarks on the Program and Applications

As it stands, the program is of more academic than practical interest. This is primarily because of the limited size of the networks which can be accommodated. A store which could hold a network of several thousand nodes, say, would be a powerful tool for research in several interesting fields. Practically, however, there is no restriction on the magnitude of the branch values in the present program and, furthermore, the speed of the program is independent of these magnitudes so that the program may find some application, albeit limited.

As examples of fields where programs of this type may find application, we mention the following:

(1) It is not difficult to visualize each of the public services—fire brigades, ambulance services and police—in a congested and extensive city of the not so distant future, having a computer in which is stored a network representation of their city (as described in the section on representation of networks in computers) and in which the branch values, interpreted as transit times, are continually changing as a result of automatic traffic monitoring on all roads within the city. A driver in an emergency, requiring to know the quickest route from $A$ to $B$, will then delay his departure for a minute or so in order to run the Moore's algorithm on the computer, and in this way save precious minutes waiting for traffic lights to change and avoiding undue concentrations of traffic.

(2) In the indexing of large libraries, cross-references may be represented as a network in which the nodes are the works themselves and the branch values are the relative availabilities of books. This is a special illustration of a much more general concept. A computer may then be used for finding perhaps the most satisfactory sequence of references relating to a particular topic.

(3) In the automatic programming of a large numerical calculation, the order in which operations are carried out is highly important in respect of computer time. Such a sequence may be representable as a path through a network in which the nodes are the fundamental operations and the branch values are the length of these operations in computer time. This is rather a special case in that branch values at any stage will be a function of the chosen path at that stage.

(4) Modern computers require wiring of considerable complexity. The economization of material is a real problem in the design of these machines and is likely to become even more important as the number of components increases, in the trend towards more powerful machines. The economization of equipotential wiring can be achieved by the construction of the minimum spanning subtree of the complete graph of equipotential terminals. The branch values are simply the lengths of the connections. With a more sophisticated interpretation of the branch values, the method may be capable of extension to a more general class of terminals.

## Acknowledgements

The work described was carried out in the Computation Laboratory of the University of Southampton,

the author's position there being made possible through a Ministry of Supply contract for which he is grateful.

More specifically, the author desires to express his thanks to Professor Rado of Reading University for stimulating his interest in network problems of this kind via an entertaining lecture delivered in March 1959 to the Mathematical Society of Southampton University. He would also like to thank Dr. G. N. Lance, the Director of the Computation Laboratory, for suggesting the paper in the first place and for his assistance, and the referee for several useful remarks.

## References

DIJKSTRA, E. W. (1959). "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, Vol. 1, No. 5, p. 269.

HOFFMAN, W., and PAVLEY, R. (1959). "A Method for the Solution of the *N*th Best Path Problem," *J. Assoc. Comp. Mach.*, Vol. 6, No. 4, p. 506.

KÖNIG, D. (1950). *Theorie der endlichen und unendlichen Graphen*, Chelsea Publishing Company.

LOBERMAN, H., and WEINBERGER, A. (1957). "Formal Procedures for Connecting Terminals with a Minimum Total Wire Length," *J. Assoc. Comp. Mach.*, Vol. 4, p. 428.

MOORE, E. F. (1957). "The Shortest Path Through a Maze." (A paper presented to the International Symposium on the Theory of Switching at Harvard University.)

PRÜFER, H. (1918). "Neues Beweis eines Satzes über Permutationen," *Arch. Math. Phys.*, 3, Vol. 27, pp. 142–144.

WEINBERGER, A., and LOBERMAN, H. (1957). "Symbolic Designations for Electrical Connections," *J. Assoc. Comp. Mach.*, Vol. 4, p. 420.

WHITNEY, H. (1932). "Non-Separable and Planar Graphs," *Trans. Amer. Math. Soc.*, Vol. 34, p. 339.