COMPUTERIZED SYSTEM RELIABILITY: SIMPLIFIED TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM (STRAP1): MINIMUM PATH METHOD (MPM)

By

DAVID VICTOR FISCHER

Bachelor of Science

Oklahoma State University

Stillwater, OK

1981

Submitted to the Graduate Faculty of the
Department of Management
College of Business Administration
Oklahoma State University
in partial fulfillment of
the requirements for the Degree of
MASTER OF BUSINESS ADMINISTRATION
May 1984

CONTENTS

COMPUTERIZED SYSTEM RELIABILITY: SIMPLIFIED TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM (STRAP1): MINIMUM PATH METHOD (MPM)

Report Approved:

Mitchell O. Locks

Advisor

[illegible]

Director of Graduate Studies

[illegible]

Head, Department of Management

Name: David Victor Fischer   Date of Degree: May 1984

Institution: Oklahoma State University

Location: Stillwater, Oklahoma

Title of Study: Computerized System Reliability:
Simplified Topological Reliability Analysis Program (STRAP1):
Minimum Path Method (MPM)

Pages in Study: 57   Candidate for Degree of Master of Business Administration

Major Field: Business Administration

0. ABSTRACT

In 1978 Satyanarayana and Prabhaker (S&P) published a paper on topological reliability analysis for single source-to-single terminal networks as an alternative to classical inclusion-exclusion. The method uses a tree-search technique to develop a system reliability formula as a function of the component reliabilities. A shortcut, nested and factored system reliability formula is generated that is exactly equivalent to the inclusion-exclusion polynomial, but is much more efficient computationally. A FORTRAN program called the Topological Reliability Analysis Program (TRAP) was also presented in the archive documentation to support this project.

In 1983, C.C. Bolaki at Oklahoma State University prepared a PL/1 structured version of the topological technique, called the Structured Topological Reliability Analysis Program (STRAP). That program stored the complete search tree before generating the equation.

In this paper, we document a new PL/1 computer program, STRAP1, that performs the same functions as STRAP, but is considerably shorter and runs faster on mainframe. This program differs from STRAP in that the equation generation and search-tree development are performed simultaneously so that the complete search tree is not stored. The result is a more efficient program. STRAP1 is a one-pass procedure which does not use the shortcut formula since this formula would be inefficient with this method.

A second program is provided which is a substitute for the existing inclusion-exclusion program, MAPS. This program is named MPM.

ADVISOR'S APPROVAL [illegible]

GLOSSARY

Acyclic Graph - a graph which does not contain a cycle.

Cycle - a path which starts and ends at the same vertex without passing throught either the start or the terminal.

Cyclic Graph - a graph which contains a cycle.

Edge - a connection between two vertices. An edge must have a vertex at each end.

Graph - a system of linked vertices.

In-degree - the number of edges which enter the vertex.

Leaf - ending node of a tree; out-degree zero.

Neutral Sequence - a sequence that can be deleted from a p-acyclic graph, with the resulting subgraph remaining p-acyclic.

Out-degree - the number of edges which leave the vertex.

p-acyclic graph - a p-graph which contains no cycles.

p-graph - a graph in which all edges lie on a path from start to terminal.

Root - the beginning node of a tree; in-degree zero.

Sequence - any one way string of edges in which all internal vertices have in-degree and out-degree one.

Start - the beginning point of the graph.

Terminal - the ending point of the graph.

Tree - a rooted graph of nodes and internodes such that the root has in-degree of zero, all other nodes have in-degree of one. Leaves have out-degree zero, all other nodes have out-degree greater than zero.

Vertex - an ending point for an edge. Any edge must have two vertices, one on each end, which it connects.

COMPUTERIZED SYSTEM RELIABILITY: TOPOLOGICAL RELIABILITY: A SIMPLIFIED VERSION (STRAP1) MINIMUM PATH METHOD (MPM)

1. Introduction

Reliability is the chance that a link or member will succeed. This chance is stated as a percentage of the total possible alternatives. The subject of this paper is to examine the methods and algorithms dealing with the reliabilities. These methods will be used in the development of computer programs which use these algorithms. These methods deal not only with component reliabilities, but also with these components as a system. Such a system is assembled into a reliability graph.

In reliability graphs, the edges are components and the vertices are assumed to be perfectly reliable. (Methods exist which allow for unreliable vertices, but these will not be assessed in this paper.) The objective is to estimate the reliability of the system as a function of the reliabilities of the components. This is done by forming a system reliability function where all component reliabilities are represented.

The classical method is inclusion-exclusion. This is proved by a set-theoretic argument called Poincare's Theorem[5]. Terms of the formula are developed from the reliabilities of the graph's components. (Based upon the work of Burris[8]). As these terms are developed, there is extensive cancellation of terms. This is caused by identical terms with opposite signs which are generated. These terms are based upon finding the minimal paths or minimal cuts. This leads to the maximum number of terms as:

$$2^m - 1$$

where m is the number of minimal paths or minimal cuts. There are usually far less terms than this.

The terms are found by finding all combinations or unions of the minimal cuts. An even formation is a term which is the union of an even number of cuts. An odd formation contains an odd number of cuts. The domination of a term is the number of even formations of the term minus the number of odd formations.

Satyanarayana and Prabhaker showed that the classical inclusion-exclusion formula is equivalent to a noncancelling graph structure[1]. They use a p-acyclic graph, a graph resulting from the union of minimal paths but which does not contain a cycle, as the fundamental term in a restructuring of the subgraphs of the reliability graph. The noncancelling terms, those with dominations not equal to zero, of the inclusion-exclusion are equivalent 1:1 to the p-acyclic subgraphs of the reliability graph. The cancelling terms are shown to be the cyclic subgraphs. The proof is somewhat incomplete but an alternate proof of this result was given by Willie in 1980[2]. Willie shows that all cyclic subgraphs, or terms from these subgraphs, will have a domination of zero. (The number of odd formations is equal to the number of even formations.) Satyanarayana and Prabhaker use a tree search to search for the p-acyclic subgraphs. They use a depth-first-search tree which was first proposed by Tarjan in 1972[7]. This algorithm provides an efficient method of graph-tree search. The search tree is constructed so that the nodes of the tree are the subgraphs of the original graph, and the internodes are the edges which must be deleted to obtain these subgraphs. The search tree is built by means of four rules which cover decycling, removing unnecessary edges, and processing terms corresponding to the noncancelling terms of the formula (section 5.2). This paper provides two computer programs which are constructed around the inclusion-exclusion method(MPM) and the topological reliability method(STRAP1).

These programs were written in PL/1 for several reasons. PL/1 allows a process of dynamic allocation which is not readily availible in many other computer languages. It also provides an excellent combination of number-crunching ability and string manipulation which was necessary for this program. Because of these abilities, the programs are not limited in the size of problem which they can process but rather are only limited by the capacity of the computer upon which they are run. However, because of the nature of inclusion-exclusion calculations, the topological reliability method will run faster for larger problems. (This may not be the case for small problems where the number of terms in the two calculations is approximately equal.) The programs are constructed in such a way that the procedure is a single-pass rather than a double-pass method so that the reliability is calculated during the tree search. The inclusion-exclusion method has been computerized before in a program called the Method for Analysis of Probabilities of Systems (MAPS)[8]. MAPS does not find the minimal paths; the user is required to input these as data. A new program is presented in this paper (section 4.5) which is a substitute for MAPS.

In addition to the topological reliability method, this program calculates the importance of each edges in the system. This is accomplished by differentiating the final inclusion-exclusion equation with respect to the reliability of the requested edge. The number provided gives the relative importance with respect to all other edges for an importance comparison. (The greater the number, the greater the importance with respect to the other edges.)

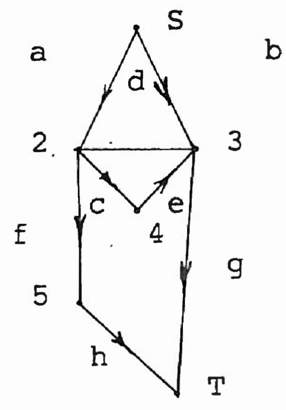
2. Graph Theory

2.1 The Graph Array

A graph is a set of connected edges and vertices. Such a system can be represented by an array in which the coordinates are the beginning and ending vertices of the given edge. For example, a directed edge which exists between vertices 3 and 4 will occupy cell A(3,4) in the array A containing the graph, a manner in which the graph can be easily manipulated by the computer. With this notation there exists only one problem. What if two edges exist between the same two vertices and are directed in the same direction. Two such edges would be parallel and uni-directional. Such a situation can be resolved by creating one new edge from these two previous edges. The new edge would occupy the same cell in the array with a new reliability of r1 + r2 - r1*r2. If the edge exists between the two vertices but is undirected, a new edge is created parallel to the first such that the two edges have opposite directions but identical reliabilities. Such a situation will not hinder inputing the edges into the graph since the new edge will not occupy the same cell as the first edge.

2.2 An Example

An example of such a graph entered into an array is as follows:



Start = 1 Terminal = 6

Fig. 1 A Sample Graph

edge	from	to
a	1	2
b	1	3
c	2	4
d	3	2
d	2	3
e	4	3
f	2	5
g	3	6
h	5	6

	1	2	3	4	5	6
1		a	b			
2			d	c	f	
3		d				g
4			e			
5						h
6						

Fig. 2 The MATRX Array

2.3 The Number of Edges and Vertices

This section is a method for counting the number of nodes and edges in a p-graph. In the array MATRX, the rows represent where the edge is coming from, the outdegree, and the columns represent where the edge is going to, the indegree (Fig. 2). The number of entries in a row will then represent the outdegree for that vertex and the number of entries in a column will represent the indegree for that vertex. The number of entries in each row and column are counted and stored in the zero cell for that row or column. The graph represented by MATRX is a directed graph (i.e. no undirected edges).

In a p-graph, all internal vertices have indegree and outdegree greater than zero. Therefore, since each edge must contribute to the indegree or outdegree of a vertex, the sum of the indegrees of all vertices will equal the sum of the outdegrees of all vertices and will be equal to the number of edges in the graph. The start vertex must have indegree zero and the terminal vertex must have outdegree zero, in a p-graph. The number of vertices may be obtained by counting the vertices which have indegree greater than zero, or counting the verticies that have outdegree greater than zero and adding one for the start or terminal.

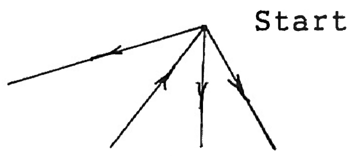
2.4 p-graphs

A p-graph is a graph in which all edges lie upon a path from the start vertex to the terminal vertex. This definition may be summarized by three conditions in an acyclic graph. If the acyclic graph meets these conditions it will be a p-graph. These conditions are:

1. The start vertex must have indegree equal zero and outdegree greater than zero.
2. The terminal vertex must have indegree greater than zero and outdegree equal zero;
3. All internal vertices must have indegree and outdegree greater than zero.

2.4.1 Conditions

The first condition concerns the start vertex. If any edge entered the start vertex, the start vertex would have an indegree greater than 0. That edge cannot be on a path from start to terminal. The condition that the start must have outdegree greater than zero is to eliminate the possibility of an incomplete graph, one which does not contain either a start or a terminal.



In-degree = 1
Outdegree = 3

Fig. 3 The Start Vertex

The second condition is analogous to the first but pertaining to the terminal.

In-degree = 3
Outdegree = 1

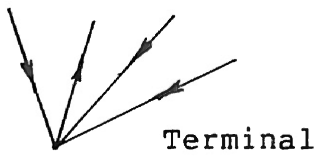
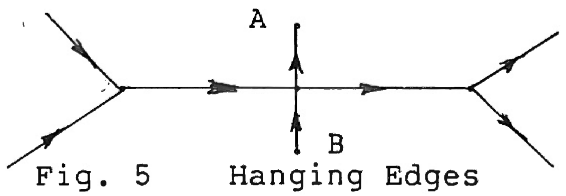


Fig. 4 The Terminal Vertex

The third condition eliminates the possibility of "hanging edges" or edges which do not have an edge leaving their ending vertex. This condition also applies to edges which do not have an edge entering their begining vertex.



Outdegree(A) = 0

In-degree(B) = 0

Fig. 5 Hanging Edges

Such edges contribute nothing to the reliability and must be eliminated.

2.4.2 Unnecessary Edges

All edges which enter the start vertex will contribute to the indegree of the start and will thus be in the start column of MATRX. All edges which leave the terminal vertex will contribute to the outdegree of the terminal and will be in the terminal column. All edges which start and end on a single vertex will contribute both an indegree and an outdegree to that vertex and will thus lie on the diagonal of MATRX. All such edges can thus be found very easily by use of MATRX. In order to determine whether the start has an outdegree and the terminal has an indegree it is only necessary to check the MATRX(start,0) cell and the MATRX(0,terminal) cell. These cells must be greater than zero to avoid an incomplete graph.

2.4.3 Deleting Edges

The topological approach requires the formation of subgraphs of an original graph. This is accomplished by systematically deleting edges under the control of the depth-first-search tree. Edges are deleted by changing the sign of the edge where it appears in the MATRX array. All edges sign unspecified, are in the subgraph, and those with minus values are those which have been deleted. This is convenient because it leaves the original graph undisturbed except for the change of sign.

2.4.4 Sequences and Neutral Sequences

Satyanarayana and Prabhaker differ in method from the method used in this paper. They delete single edges in the first two rules of their method. However, if a single edge is deleted from a sequence, it will always leave a 'hanging' edge which must be deleted in subsequent steps in order to achieve a p-acyclic graph. Therefore, in this proceedure, single edges are deleted alone only if they are not a part of a larger sequence. When an edge is contained in a sequence, all other edges which comprise the sequence are deleted with it.

When a sequence is deleted, that sequence will be called by an edge within that sequence. When an edge is deleted, the contribution to the indegree or outdegree of the bounding vertices must also be deleted. When an edge is deleted from a sequence, the deletion leaves vertices which have either indegree or outdegree of zero. Furthermore, these vertices will be one, or both, of the bounding vertices of the deleted edge. Therefore, if the edge contained in MATRX(v1,v2) is deleted and is a part of a sequence, either or both of cells MATRX(0,v1) or MATRX(v2,0) will be zero. For example:

The given sequence

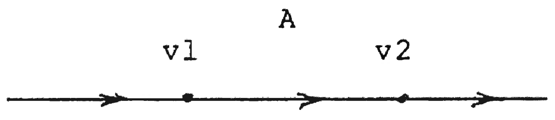


Fig. 6 An Edge within a Sequence

If the given edge, A, is deleted, the remainder of the sequence must also be deleted. If A is deleted, the contribution to the outdegree of v1 and the indegree of v2 will also be deleted. Vertex v1 will now have an outdegree of zero or MATRX(0,v1) = 0 and v2 will now have an indegree of zero or MATRX(v2,0) = 0. This is easy to check and such an occurrence will show that the edges on either side of A are a part of the sequence containing A.

A special type of sequence is discussed by Satyanarayana and Prabhaker. When a sequence can be deleted from a p-acyclic graph and the resulting subgraph will be p-acyclic, such a sequence is refered to as a neutral sequence. This type of sequence is used in the processing of p-acyclic subgraphs with the topological approach.

3. Search Tree

3.1 The Search Tree

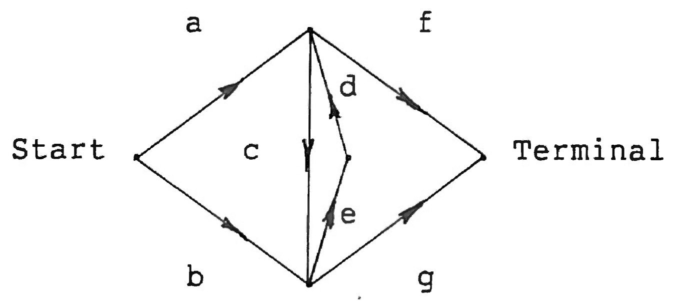
A tree is a rooted network of nodes and internodes in which the root has indegree of zero and all other nodes have indegree of one. The search tree is a means of recursing the tree in a systematic manner. The primary terminology of a search tree is as follows:

1. Node - a connection between branches of the tree.
2. Branches - the internodes. All branches must be bounded by two nodes.

3. Children - internodes which contribute to the outdegree of a given node.
4. Father - internode which contributes to the indegree of a given node.
5. Ancestors - any internodes which preceeds a given node.
6. Elder Brothers - any children to the right of the given child.
7. Younger Brothers - any children to the left of the given child.

3.2 Depth-First-Search

The depth-first-search tree is constructed by finding all children of the given nodes and then proceeding to the eldest child. Each internode represents the deletion of a sequence. The eldest child is then processed in a similar manner by finding all its children and proceeding to the eldest child. When a leaf, the bottom-most node with outdegree of zero, is reached the procedure backtracks. Backtracking continues until a node is encountered which has a child which has not been visited. The procedure then visits the eldest child which has not yet been visited. In the following example, the search tree is simplified to include sequences instead of edges. In this search tree, the nodes representing p-acyclic subgraphs have the edges contained in the search tree beside these nodes.



The search tree would appear as:

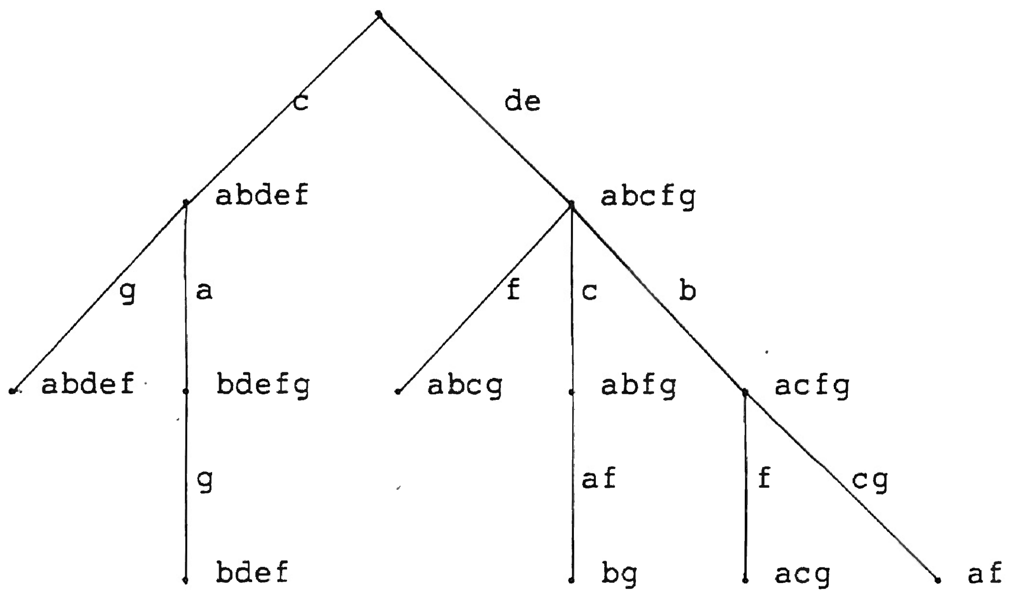


Fig. 7 A Sample Search Tree

4. Inclusion Exclusion

4.1 The Method

Inclusion exclusion (IE) is a method by which the reliability of a graph is found by finding all combinations of the minimal paths. A minimal path is a sequence which begins at the start vertex and ends at the terminal vertex.

This sequence may not be a sequence in the original graph but is a subgraph of that original graph. The first path and the second path are combined to create a new subgraph. The third path is then combined in all combinations with the first two to create three new paths, and so on. Since all possible graphs are combinations of minimal paths, all graphs must have all edges on a path from start to finish. The number of subgraphs which will be generated is equal to two raised to the number of minimal paths (2**N1 - 1). Since this number will increase exponentially with an increase in the number of minimal paths, this method is obviously not suited for large problems.

4.2 Reliability

The reliability terms are calculated by multiplying the reliabilities of all edges contained in the generated subgraph together (the terms of the formula are 1:1 with the subgraphs; however, many of these terms will cancel) and multiplying by -1 to the Q power. In this case, Q is the number of minimal paths which were combined to obtain the subgraph plus one (Q = N1 + 1).

$$R * -1^{Q}$$

where R is the multiplication of the reliabilities of the edges contained in the subgraph. The terms are then added together to find the total reliability of the graph.

4.3 A Method of Finding Combinations

If the number of minimal path is large, the process of finding all combinations can become difficult. As a result, a simple algorithm has be created to find all minimal paths.

The inclusion-exclusion process finds all combinations of the minimal paths or minimal cuts. The result is a subgraph which contains some or all of the edges of the original graph. The subgraph yields a term of the reliability equation with an appropriate sign. This sign is a function of the number of cuts which were combined to create the subgraph.

This algorithm finds all combinations of the cuts by either adding or deleting a cut from the current combination. If the number of cuts in the combination is odd, the sign of the term will be positive and if the number is even, the sign will be negative. The number of cuts in the combination will always change by one. It is no longer necessary to calculate the sign of the term but only to change each time the series is changed.

A series is created which has maximum length N1 (the number of minimal paths). The series is initially empty and the first cell is established as one. The series now has length one. If the last number of the series has value other than N1, a new value is added to the end of the series. This value is one greater than the previous value. If the value of the last number is equal to N1, this element is deleted and 1 is added to the previous number. This will create a series which is as follows:

```
1                      N1 = 4
1 2
1 2 3
1 2 3 4
1 2 4
1 3
1 3 4
1 4
2
2 3
2 3 4
2 4
3
3 4
4
```

Fig. 8 The Series

When the series has a length of zero, the algorithm stops. This algorithm will give all combinations of paths with the exception of the null set. In the example, the first combination will consist of only the first minimal path; the second combination will consist of the first and second; the third combination will consist of the first, second, and third, etc.

4.4 The Minimum Path Method (MPM)

The Minimum Path Method finds not only all possible unions of the minimal paths or minimal cuts, but also finds the paths themselves. Since this is slightly different than the usual inclusion-exclusion procedure, the program will be called the Minimum Path Method (MPM). This program is shown in appendix B.

4.4.1 Minimal Path Search

In MAPS, it was necessary to input the minimal paths; but with the use of the array MATRX, it now becomes easy to obtain these minimal paths by search. In order to find minimal paths, it is necessary to start with the start vertex and search the graph from this point. The graph search then proceeds by traversing the start row until a value greater than zero is found. This search is a recursive search with the value found in the previous search being used to find the value in the current search. The row in which that value is found is the new vertex, and the search has progressed from the start vertex (vertex 1 in Fig. 1), to the next vertex that can be visited from the start vertex (The lowest numbered vertex, vertex 2 in Fig. 1). If there are many vertices which may be visited from the start vertex, The vertex with the lowest number will be the first vertex visited. Since the rows give the outdegree of the vertices and the columns give the indegree of the vertices, when a cell is found with a value greater than zero (all empty cells are assigned a value of zero), the process has found the contribution to the outdegree of that vertex by the edge. If there is an outdegree for the given edge then the edge must also contribute to the indegree of one, and only one, vertex. The indegree is found in the columns. It is only necessary to read the column number to see which vertex the edge contributes indegree to. In order to search the graph, the process passes from edge to vertex to edge to vertex. When the vertex is found that the edge enters, it is only necessary to find an edge which leaves, or contributes to the outdegree of, the vertex to continue the search. This may be found in the row coresponding to the vertex. For example, let us consider the matrix in Fig. 2:

This is the same matrix that was examined earlier.

N = 6 M = 9

	a	b			
		d	c	f	
	d				g
		e			
					h

Fig. 9 MATRX

The search progresses by beginning with the start row, in this case the first row, and searching until a value is found which is greater than zero. As the vertices are visited, they are stored so that they cannot be visited again. This eliminates the possibility of a cycle since a cycle must visit the same vertex twice. The search will stop at 'a'. The edge 'a' is found in the second column so the search will start again in the second row. The search progresses until a cell is found which contains a value greater than zero. The search progresses to 'd'. The search then starts in the third row. This time, a value is found in the second column but vertex two has been previously visited. The search will continue to 'g'. The edge 'g' is found in the sixth row which is the terminal column (Terminal = 6). A minimum path has been found (a-d-g).

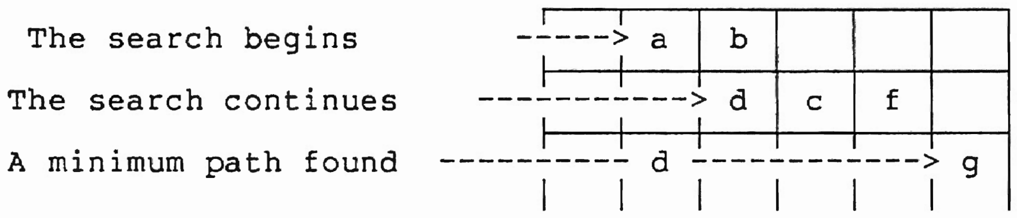


Fig. 10 One Minimal Path Search

The search then backtracks to 'd' in the second row and the search continues to 'c'. The search finds 'e' in the fourth row and 'd' in the third row. Vertex two has been previously visited ('a') so the search continues to 'g' which gives another minimal path (a-c-e-g). The search backtracks to 'c' and continues to 'f'. In the fifth row, the proceedure finds 'h' in the terminal column, another minimal path (a-f-h). When the search has backtracked to the start row, it will eventually traverse the row. This will signal the ending of the search. At this point, all of the minimal paths will have been found.

4.5 The Program

4.5.1 Data Input

The data may be input in the same manner for the two programs and the same data files may be used for either program. A sample data file may be seen in Appendix C Before the data input begins, any comments that the user may wish to input may be entered. The program will echo print these comments before the procedure starts. The only restriction on these comments is that there may not be more than ten blank lines at one time or the program will stop. This is to ensure that the program is not in a continuous loop.

Processing is started by a card as follows:

```
$JOB
```

The $JOB must be in columns one through four and be in capital letters. Since the program will only read the first four characters of this card, anything may be written on the remainder. The program then reads the number of vertices in the graph, the number of edges to be read from the data, the number of the start vertex, and the number to be read from the terminal vertex. Next, the edge data is input as the beginning vertex for the given edge, the ending vertex, the directed status (one equals directed and zero equals undirected), and the reliability of the given edge. All data is read in as stream so there is no need to input the data on separate cards unless it is necessary for clarity. The data must be separated by blanks. It is not necessary to input the number of the edge as the program will number the edges in the order in which they are input.

The program must call the data file with a Data Definition card (DD) as follows:

```
//B14805A JOB (14805    . . .    the standard JCL cards
        .
        .
        .
// EXEC PLC,REGION=500K
//INPUT DD DSN=U14805A.SAMPLE.DATA(C),DISP=SHR
//SYSIN DD *
*PL/C NOSOURCE TIME=(,5) PAGES=200
```

The region will have to be increased for larger problems. The data file in this case has been specified as:

```
     U14805A.SAMPLE.DATA(C)
```

The NOSOURCE option will suppress a listing of the program.

4.5.2 Minimum Path Search

The search is conducted as has been described with the vertices which have been visited being stored in the array UNA (Unavailable). The minimum paths are stored in a temporary array called PATH and are transferred to a permanant array called PATHS. The new algorithm is then applied to find the terms which are then printed.

5. Topological Reliability

5.1 The Search Tree

The search tree is built by deleting edges from the given graph to produce a new subgraph. The determination of which edges to delete were found by means of the rules. In this particular search tree, the nodes are designated as the resulting subgraphs and the internodes are designated as the sequences which are deleted to find that subgraph.

5.2 The Rules

5.2.1 Rule One

Rule one is called if the subgraph is cyclic. This rule breaks all cycles by deleting one sequence of each cycle. The sequences of the cycle then become the children of the subgraph. If the resulting graph is cyclic, rule one is called again. If it is acyclic, rule two is called. Finally, if the resulting graph is incomplete, the procedure backtracks. (Backtracking is equivalent to proceeding up the tree instead of down it; therefore, since moving down the tree is accomplished by deleting edges from the subgraph, moving up the tree is accomplished by restoring the same edges which have been deleted.)

When the procedure backtracks, the deleted sequence of the cycle is restored and the next sequence is deleted until all sequences of the cycle have been deleted.

5.2.2 Rule Two

Rule two deletes all unnecessary edges such as edges which do not lie on a path from start to terminal. If the resulting graph is not a p-graph or is an incomplete graph, then the procedure backtracks. Otherwise, the graph must be p-acyclic and rule three is called.

5.2.3 Rule Three

Rule three is called if the subgraph is p-acyclic but does not have a p-acyclic father. Rule three finds all sequences which can be deleted from the subgraph and the resulting graph will remain p-acyclic. All such sequences then become the children of that particular graph. Rule four is called.

5.2.4 Rule Four

Rule four is called when the given graph is p-acyclic and has a p-acyclic father. Rule four states that all children of the given graph are equal to the younger brothers of the father. Thus, the sequences which are to the left of the sequence which was deleted to obtain the given graph are then deleted to find new children of the given graph. Rule four is then called again until there are no younger brothers of the father or until the removal of the sequence causes an incomplete or non-p-acyclic graph.

Due to the removal of other sequences, the sequence being deleted may be part of a larger sequence. In such a case, the entire larger sequence must be deleted.

5.2.5 The Weight Rule

If an edge is an elder-brother or the elder-brother of an ancestor, the edge is said to be in the weight of that edge. This means that no edge which is in the weight may be deleted from the given graph. This weight rule is applied to all of the rules. If no edge which is in the weight is deleted, there can be no duplications in the generations of subgraphs.

5.2.6 The Reliability Equation

Satyanarayana and Prabhaker developed a reliability equation which can be read directly from the search tree. This equation involves finding the product of the reliabilities of all of the edges in the original graph and dividing the deleted edges from this product as the edges are deleted. This may be done by recursing the tree after it has been built since the deleted edges to each graph are the internodes of the search tree.

5.2.7 Edge Importance

The edge importance is a measure of the importance of each individual edge. The importance is calculated by finding the linkset and taking the partial derivative of the each term with respect to the desired edge. For example:

The Linkset:

-1	A B C D
1	B C D
1	C D
-1	A C D

If Ra is the reliability of edge A, Rb the reliability of edge B, etc. then the reliability equation would appear as:

$$-RaRbRcRd + RbRcRd + RcRd + RaRcRd$$

If the partial derivative were taken with respect to B, the importance of B would be:

$$-RaRcRd + RcRd$$

More simply, any term which contains B will now not contain B; if the term does not originally contain B, it will not appear in the importance equation. This causes importance to be dependant upon whether it appears in the terms as well as the reliabilities of the other edges.

5.3 Simplified Topological Reliability

5.3.1 Efficiency

There are several things which may be done to improve the efficiency of the topological reliability procedure. One is to make the original graph a p-graph before the procedure begins. This involves removing all unnecessary edges at the beginning. These edges include:

1. Hanging edges.
2. Edges which start and terminate at the same vertex.
3. Edges which either enter the start vertex or leave the terminal vertex.

5.3.2 The Reliability Equation

In order to improve the efficiency, construct a program which will act like a search tree instead of one which will build a search tree. This eliminates the need to recurse the tree a second time and even eliminates the need to store the search tree. After the program has processed a branch of the tree, it is no longer needed and can be discarded. The question then arises as to whether the equation will prove more efficient than adding the reliabilities of each term (the linkset). In order to use the equation, it will either be necessary to add terms to the equation as the tree is processed, which will include those terms which are not p-acyclic (meeaning that those terms will have to be removed from the equation during backtracking), or it will be necessary to store the tree, which can become very large even for a simple graph, and recurse the tree after it is built. Either choice will involve extra calculations which are unnecessary. In order to solve the equation, the number of actual calculations which will be necessary on most graphs, multiplications, divisions, or additions, is essentially equal for both the single term method and the equation.

5.3.3 Which is Best?

The method of finding the linkset, the set of individual subgraphs, has some value since it shows exactly which terms are in the reliability formulation. (The linkset is automatically generated during processing.) This makes the checking of the results easy since duplications can be spotted relatively simply. Checking the equation, except against an established answer, is tedious and somewhat fruitless. The equation has one advantage in that the search tree, if it is desired, can be quickly found by building it from the equation. However, it would be relatively simple to have the program print the edges as it deletes and restores them. For these reasons, it is somewhat doubtful whether the equation is of any value; therefore, it has not been included in the program given in this paper. This program finds the linkset by using the four rules given by Satyanarayana and Prabhaker and then finds the total reliability of the system. The program in addition finds the importance of the edges which are not easily obtainable from the equation. In any case, the finding of the reliability is a small portion of the time required to complete the analysis of the system. The major part of the time is spent developing the search tree.

5.4 STRAP1

5.4.1 Startup

The program (Appendix A) begins by inputing the number of vertices and the number of edges which will be read from the data file. The program next inputs the starting vertex and the terminal vertex. The number of vertices will not change throughout the program, unlike the number of edges. The edges may be undirected in the input data which will cause the program to create a new edge parallel to the original but in the opposite direction. This will cause two directed edges to be input instead of one undirected.

5.4.2 Super Edges

The program next checks for super edges which will indicate a multiple source and/or a multiple terminal problem[3]. This does not change the method of working the problem but simply causes the super edges to be put initially into the weight so that they cannot be deleted during processing[4]. The program recognizes super edges when it finds a negative sign in front of the reliability. Reliability must be a positive value so in this case the negative sign simply acts as a flag. This will mean a reliability of minus one since a super edge must be perfectly reliable.

5.4.3 Array Size

This method of startup causes a minimum of necessary memory to be used by allowing the program to size its array according to what is necessary. Arrays are sized according to either the number of vertices or the number of edges. Those using the number of vertices are sized before data input and those using the number of edges are sized after. The startup is completed by removing the unnecessary edges to create a starting p-graph.

5.4.4 Rule One

Rule one starts by calling a subroutine within it called SEARCH. SEARCH traverses the graph much the same way that the minimum paths were found except that when SEARCH encounters a vertex which has been previously visited, it recognizes this as a cycle. During the search process, SEARCH keeps track of the vertices that have been visited both is UNA (Unavailable) and in VERTX (the vertices in the order in which they have been visited). By knowing the order in which the vertices have been visited, when SEARCH finds a cycle, visits a vertex which has been previously visited, the cycle can be found by means of the previous nodes. The array VERTX is traversed from the first cell until the current vertex is found. All vertices between these two vertices will be vertices in the cycle. This vertex and the vertex in the next cell will be the beginning and ending vertices of the first edge of the cycle. This edge is deleted by calling REMOVE and RULE_ONE is called again recursively. The act of deleting the edge of the cycle may disturb the previous search. By calling RULE_ONE, the search of the graph is initialized with this edge deleted thus starting the search over with a new subgraph. PLI creates 'environments' in which the program operates. By calling RULE_ONE recursively, the program creates a 'sub-environment' in which to process the new subgraph. When the program returns, it will have the same parameters which existed at the time of the call. This means that the current search will still be in progress. The edge which was deleted can be restored and the next edge can be deleted with a new call to RULE_ONE. Because of the nature of a recursive procedure, the program can act exactly like the search tree without having to build and store the tree. The tree deletes edges, or sequences, and checks to see what type of subgraph results. The tree then calls the appropriate rule. This is exactly what this program does. In this program, calling a rule is analogous to going down the tree and a return is analogous to backtracking. When RULE_ONE is able to search the entire tree without finding a cycle, it calls RULE_TWO.

5.4.5 Rule Two

Since RULE_TWO can only be reached by first going through RULE_ONE, the subgraph must be acyclic. As has been previously stated, if an acyclic graph has all internal vertices with indegree and outdegree greater than zero it will be p-acyclic. This is true because in order for every vertex to have an indegree and an outdegree, it must have an edge on either side of it. Since an edge must have a vertex on either end, that vertex must also have an edge on either side of it. The only way that one vertex can have an indegree and an outdegree without entering the terminal or leaving the start, is for a cycle to both enter and leave that vertex. The given subgraph may not be acyclic because the deletion of edges may cause a portion of the graph, which may contain a cycle, to be unreachable by search. During SEARCH, the edges which are traversed during the search are put in an array called USED. If there are any edges in the graph which have not been searched, and are not in USED, they will be deleted. If any of these edges are in the weight, the procedure returns since the subgraph cannot be made p-acyclic. All other edges must be 'hanging edges' and will have an indegree or outdegree of one of their bounding vertices equal to zero If any of these is in the weight, the procedure returns, RULE_TWO then calls P_MATRX which checks the subgraph to see if it is p-acyclic. If the subgraph is p-acyclic, RULE_THREE is called.

5.4.6 Rule Three

Rule three searches MATRX for entries which are greater than zero ane deletes these entries one at a time. When the edge is deleted, it calls P_MATRX to check for a p-acyclic graph. If the graph is p-acyclic, the edges, along with all edges which are in a sequence with it, are stored in CHILDREN. The edge is then restored, along with all other edges in the sequence, and the next entry is deleted. This finds all neutral sequences of the given graph. Since the current graph is p-acyclic, RELIABILITY is called. Once all neutral sequences are found, rule four is called and the array CHILDREN is passed to it.

5.4.7 Rule Four

Rule four takes the array CHILDREN and searches this array for an entry greater than zero. This entry will be the begining vertex of the child. Rule four then searches this column of MATRX for the entry I. Rule four then deletes this entry, sets CHILDREN(I) to zero, and calls itself recursively. When rule four returns it will continue the search from this point in the array CHILDREN.

6. Data Input

This paper includes two different methods of finding graph reliability and a program for each. The data input for each program is identical. The data includes provision for comments to be entered at the first of each problem. These comments may be anything that the user wishes and may be as long as is necessary. The program will simply output these comments without storing them. When the user is through with these comments, the data input will be initiated by a card as follows:

```
$JOB
```

This card must start in the first column and must be in capital letters.

6.1 The Data

The data follows immediately after the $JOB card and is in the following order: the number of vertices which the graph contains, the number of edges to be read by the program, the number of the start vertex, the number of the terminal vertex, the edge data.

The edge data contains: the beginning vertex of the edge, the ending vertex of the edge, the directed status (zero for undirected and one for directed), and the reliability of the edge. These four items must be given in this order for all edges, the number of which was given earlier. It is not necessary to input the number of the edge since the program will number the edges in the order in which they are input.

6.1.1 Topological Reliability

The topological reliability program provides for options which may be input into the program for the desired result. These options are input on the $JOB card with one space between options. These options are: [DATA] to echo print the data. [MATRIX] to print the MATRX matrix. This option may not be used except with the DATA option and will automatically cause the data to be printed. [LINKSET] to print the linkset. [IMPORTANCE] to print the edge importances. [TRACE] to print the edges as they are removed and restored, and the P_FLAG status as it is checked. ('1' is a p-graph.) This option will cause the page counter not to be reset at the top of each page so that the heading will not always appear at the top of each page. This option also slows the program down considerably and should not be used for large problems.

These options all slow the program down and thus decrease the efficiency.

TOPOLOGICAL RELIABILITY

RULES

1. Rule One - Breaks all cycles, one sequence at a time, by deleting them from the graph. This creates subgraphs which are the children of the parent graph. There are as many children as there are sequences in the cycle. The weight rule applies.
2. Rule Two - If the given graph is not a P_Graph, change it to a P_Graph by deleting unnecessary edges one at a time until the graph becomes a P_Graph. This creates a succession of children since each non-P_Graph will have only one child. This rule can be applied first to improve efficiency. The weight restriction applies but edges deleted in this rule need not be included in the weight.
3. Rule Three - p-acyclic graph with a non-p-acyclic father. Find all neutral sequences by deleting sequences individually. If the graph remains p-acyclic, this sequence defines a child. The weight rule applies.
4. Rule Four - p-acyclic graph with a p-acyclic father. All sequences deleted to find new children are confined to the sequences deleted to find the younger brothers of the father. The sequence may be lengthened to include other edges in either direction

but the weight rule is in effect. If a non-p-acyclic graph occurs from the removal of a sequence, then backtrack.

5. Weight Rule - No edge which was deleted by an elder-brother or the elder-brother of an ancestor may be deleted. This prevents duplication.

Appendix A;

The Simplified Topological Reliability Analysis Program

```
//B14805A JOB (12817,000-00-0000),CLASS=4,TIME=(4,),
// MSGCLASS=X,NOTIFY=*
/*PASSWORD ?
/*ROUTE PRINT LOCAL
/*JOBPARM ROOM=B,FORMS=9021
// EXEC PLC,REGION=500K
//INPUT DD DSN=U14805A.SAMPLE.DATA(F),DISP=SHR
//SYSIN DD *
*PL/C TIME=(4,)
1/*******************************************************************/
 /* SIMPLIFIED TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM.             */
 /* THIS PROGRAM USED S&P TRAP PROCESS IN A MORE EFFICIENT MANNER    */
 /* FOR THE COMPUTER USER.                                           */
 /* THE DATA IS INPUT IN THE FOLLOWING MANNER:                       */
 /* $JOB DATA MATRIX LINKSET IMPORTANCE TRACE                        */
 /* N M S T                                                          */
 /* B E D R                                                          */
 /*    .                                                             */
 /*    .                                                             */
 /*    .                                                             */
 /*                                                                  */
 /*     IN THE EXAMPLE ABOVE, THE BEGINNING DATA CARD STARTS WITH    */
 /* A $JOB CARD.  THIS CARD HAS OPTIONS AFTER IT WHICH CAN BE        */
 /* ENTERED IN ANY ORDER AND AS MANY OF THE OPTIONS AS DESIRED.      */
 /* THE DATA AND MATRIX OPTIONS NEED NOT BE ENTERED TOGETHER AS      */
 /* THE MATRIX OPTION INCLUDES THE DATA OPTION.                      */
 /*     THE NEXT LINE OF DATA INCLUDES THE NUMBER OF VERTICES, N     */
 /* THE NUMBER OF EDGES, M, THE NUMBER OF THE START VERTEX, S, THE */
 /* THE NUMBER OF THE TERMINAL VERTEX, T.                            */
 /* THE PROGRAM WILL NEXT READ THE NUMBER OF EDGES THAT WAS INPUT    */
 /* IN THE PREVIOUS LINE.  THESE INCLUDE THE BEGINNING NODE, THE     */
 /* ENDING NODE, THE DIRECTED STATUS (1 = DIRECTED,                  */
 /* 0 = UNDIRECTED), AND THE RELIABILITY OF THE EDGE.                */
 /* BEFORE THE $JOB CARD, ANY COMMENTS MAY BE ENTERED.               */
 /*******************************************************************/
 MAIN: PROC OPTIONS(MAIN);
   DCL (N,M,LINE) FIXED;
   DCL (C_FLAG,D_FLAG,M_FLAG,L_FLAG,I_FLAG,T_FLAG) BIT(1) INIT('0');
   DCL (COMMENTS,OPTIONS) CHAR(80) VAR;
   DCL REL FLOAT;
   ON ENDFILE(INPUT) C_FLAG = '0';
   CALL HEADER;
   GET FILE(INPUT) EDIT (COMMENTS) (COL(1),A(80));
     C_FLAG = '1';
     DO WHILE (C_FLAG);
       IF (LENGTH(COMMENTS) > 3) THEN IF (SUBSTR(COMMENTS,1,4) ¬=
                      '$JOB') THEN DO;
         PUT SKIP EDIT (COMMENTS)(A(80));
         LINE = LINE + 2;
         IF LINE > 60 THEN CALL HEADER;
         GET SKIP FILE(INPUT) EDIT (COMMENTS) (COL(1),A(80));
       END;
       ELSE C_FLAG = '0';
     END;
```

```
    C_FLAG = '1';

    COMMENTS = SUBSTR(COMMENTS,6,LENGTH(COMMENTS)-6);

    D_FLAG = '0';
    M_FLAG = '0';
    L_FLAG = '0';
    I_FLAG = '0';
    T_FLAG = '0';
    IF LINE > 53 THEN CALL HEADER;
    PUT SKIP(5) EDIT ('OPTIONS USED',COMMENTS) (A,COL(1),A(80));
    DO WHILE (COMMENTS ¬= ' ');
      OPTIONS = SUBSTR(COMMENTS,1,INDEX(COMMENTS,' '));
      COMMENTS = SUBSTR(COMMENTS,INDEX(COMMENTS,' ')+1,LENGTH(COMMENTS)
            - INDEX(COMMENTS,' '));
      IF OPTIONS = 'DATA' THEN D_FLAG = '1';
      IF OPTIONS = 'MATRIX' THEN DO;
        D_FLAG = '1';
        M_FLAG = '1';
      END;
      IF OPTIONS = 'LINKSET' THEN L_FLAG = '1';
      IF OPTIONS = 'IMPORTANCE' THEN I_FLAG = '1';
      IF OPTIONS = 'TRACE' THEN T_FLAG = '1';
    END;
  GET FILE(INPUT) LIST (N,M);
  BEGIN;
    DCL (IN,OUT,GRAPH(0:N,0:N),Z,O,MO,SIGN,N1,I,J,
         D_NUM,START,TERMINAL,LINE) FIXED;
    DCL (REL,GRAPH_REL,EDGE_REL(2*M)) FLOAT;
    GRAPH = 0;
    D_NUM = 0;
    VERTX = 0;
    REL = 1.0;
    LINE = 0;
    GRAPH_REL = 0.0;
     MO = -1;
      Z = 0;
     N1 = 1;
      O = 1;
    GET FILE(INPUT) LIST(START,TERMINAL);
    CALL HEADER;
    IF (D_FLAG) THEN
    PUT SKIP(2) EDIT ('  NO. OF VERTICIES = ',N,'  NO. OF EDGES = ',M)
                     (A,F(3),A,F(3));
    IF (D_FLAG) THEN
    PUT SKIP(2) DATA (START,TERMINAL);
    LINE = LINE + 4;
    CALL DATA_IN;
    BEGIN;
      DCL (WEIGHT(M),P_FLAG,UNA(N),BIT1,BIT0) BIT(1);
      DCL D_SEQ(M+1,2) FIXED;
      DCL IMPORTANCE(M) FLOAT;
      D_SEQ = 0;
      IMPORTANCE = 0.0;
      BIT1 = '1';
      BIT0 = '0';
      WEIGHT = BIT0;
      DO I = 1 TO M;
        IF (EDGE_REL(I) = -1) THEN WEIGHT(I) = BIT1;
        ELSE REL = REL * EDGE_REL(I);
      END;
/*******************************************************************/
/*   THE FOLLOWING ROUTINE SEARCHES THE GRAPH AND COUNTS THE NON- */
/* ZERO ENTRIES.  THE TOTAL FOR EACH ROW AND COLUMN IS THEN PUT    */
/* IN THE ZERO CELLS FOR EACH ROW AND COLUMN.                      */
/*******************************************************************/
      DO I = 1 TO N;
        DO J = 1 TO N;
          IF (GRAPH(I,J) > 0) THEN DO;
            GRAPH(I,0) = GRAPH(I,0) + 1;
            GRAPH(0,J) = GRAPH(0,J) + 1;
          END;
        END;
      END;
/*******************************************************************/
/*   THE FOLLOWING DO LOOP CHECKS TO SEE IF THERE ARE ANY EDGES    */
/* ENTERING THE STARTING NODE, LEAVING THE TERMINAL NODE, OR       */
/* WHICH LEAVE AND ENTER A SINGLE NODE.  SUCH EDGES WILL           */
/* BE IN THE START COLUMN, THE TERMINAL ROW OR THE DIAGONAL.       */
/* THESE TYPES OF EDGES WILL CONTRIBUTE NOTHING TO THE RELIABILITY*/
/* AND MUST BE REMOVED ALONG WITH ANY SEQUENCES WHICH CONTAIN      */
/* THEM.  THIS PROCEEDURE ALSO REMOVES ALL 'HANGING' EDGES.        */
/*******************************************************************/
      DO I = 1 TO N;
        IF (GRAPH(I,START) > 0) THEN CALL REMOVE(I,START,Z,BIT0);
        IF (GRAPH(TERMINAL,I) > 0) THEN CALL REMOVE(TERMINAL,I,Z,BIT0);
        IF (GRAPH(I,I) > 0) THEN CALL REMOVE(I,I,Z,BIT0);
        IF (I ¬= START) THEN IF (GRAPH(0,I)=0) THEN IF (GRAPH(I,0)>0)
          THEN DO J=1 TO N;
            IF (GRAPH(I,J)>0) THEN CALL REMOVE(I,J,Z,BIT0);
          END;
        IF (I¬=TERMINAL) THEN IF (GRAPH(I,0)=0) THEN IF (GRAPH(0,I)>0)
          THEN DO J = 1 TO N;
             IF (GRAPH(J,I)>0) THEN CALL REMOVE(J,I,Z,BIT0);
          END;
      END;
      IF (D_FLAG) THEN CALL HEADER;
      CALL RULE_ONE;
      PUT SKIP EDIT ('_________','GRAPH RELIABILITY',GRAPH_REL)
          (COL(60),A,COL(1),A,COL(60),F(8,6));
      IF (I_FLAG) THEN DO;
        CALL HEADER;
        PUT SKIP(6) EDIT('EDGE IMPORTANCE')(A);
        PUT SKIP (4);
        DO I = 1 TO M;
          PUT SKIP EDIT ('IMPORTANCE(',I,') = ',IMPORTANCE(I))
                        (A,F(2),A,F(9,6));
        END;
      END;
      GET FILE(INPUT) EDIT (COMMENTS) (COL(1),A(80));
      PUT PAGE;
1/**********************************************************************/
 /*    RULE ONE SEARCHES THE GRAPH FOR CYCLES AND DELETES THE THE      */
 /* EDGE OF THE CYCLE.  RULE ONE IS THEN CALLED AGAIN RECURSIVELY      */
 /* UNTIL THE SEARCH IS COMPLETED WITH NO CYCLES FOUND.  RULE TWO IS  */
 /* THEN CALLED.  RULE ONE KEEPS TRACK OF THE EDGES VISITED DURING     */
 /* FINAL SEARCH IN ARRAY 'USED'.  WHEN RULE TWO RETURNS, RULE ONE     */
 /* RESTORES THE FIRST EDGE OF THE MOST RECENT CYCLE AND DELETES THE  */
 /* NEXT EDGE AND CALLS RULE ONE UNTIL ALL EDGES OF THE CYCLE HAVE     */
 /* BEEN DELETED.  THE PROCEDURE RETURNS TO THE PREVIOUS CYCLE UNTIL  */
 /* ALL CYCLES HAVE BEEN PROCESSED.                                    */
 /**********************************************************************/

 RULE_ONE: PROC RECURSIVE;
   DCL (VERTX(N+1),N1) FIXED;
   DCL (USED(M),ANCESTOR(M),CYCLE_FLAG) BIT(1) INIT('0');
     VERTX = 0;
     UNA = BIT0;
     USED = BIT1;
     VERTX(1) = START;
     N1 = 1;
     CALL SEARCH(START);
     IF (CYCLE_FLAG = BIT0) THEN CALL RULE_TWO;
     RETURN;

 SEARCH: PROC(START) RECURSIVE;
   DCL (START,L,L1,N2) FIXED;
   N1=N1+1;
   IF (START = TERMINAL) THEN RETURN;
   IF (UNA(START)) THEN DO;
     CYCLE_FLAG = BIT1;
     L = 1;
     DO WHILE (VERTX(L) ¬= START);
         L = L + 1;
     END;
     DO L1 = L TO N1-2;
       IF (WEIGHT(ABS(GRAPH(VERTX(L1),VERTX(L1+1)))) = BIT0) THEN DO;
         CALL REMOVE(VERTX(L1),VERTX(L1+1),Z,BIT1);
         ANCESTOR = WEIGHT;
         CALL RULE_ONE;
         WEIGHT = ANCESTOR;
         CALL RESTORE(VERTX(L1),VERTX(L1+1));
       END;
     END;

   RETURN;
   END;

   UNA(START) = BIT1;

   DO N2= 1 TO N;
     IF (GRAPH(START,N2) > 0) THEN DO;
```

```
       VERTX(N1) = N2;
       USED(GRAPH(START,N2)) = BIT0;
       CALL SEARCH(N2);
       IF (CYCLE_FLAG) THEN RETURN;
       UNA(N2) = BIT0;
       VERTX(N1) = 0;
       N1 = N1 - 1;
     END;
   END;
   N2 = N;
   RETURN;
   END SEARCH;
1/*******************************************************************/
 /*   RULE TWO DELETES ALL UNNECESSARY EDGES BY DELETING ALL EDGES    */
 /* WHICH HAVE A BEGINNING VERTEX WITH IN-DEGREE, GRAPH(N,0), OF ZERO */
 /* OR AN ENDDING VERTEX WITH OUT-DEGREE, GRAPH(0,N), OF ZERO.  SUCH  */
 /* EDGES ARE KNOWN AS 'HANGING EDGES'.  THE RULE ALSO DELETES ALL    */
 /* EDGES WHICH WERE NOT VISITED BY THE FINAL SEARCH OF RULE ONE.     */
 /* THE RULE THEN CALLS P_GRAPH TO SEE IF THE REMAINING GRAPH IS A    */
 /* P_GRAPH.  IF SO, THE RULE CALLS RULE THREE, OTHERWISE IT RETURNS. */
 /*******************************************************************/

 RULE_TWO: PROC;
 DCL (I,J) FIXED;
   DO I=1 TO N;
     IF (I ¬= START) THEN IF (GRAPH(0,I)=0) THEN IF (GRAPH(I,0)>0) THEN
       DO J=1 TO N;
         IF (GRAPH(I,J)>0) THEN CALL REMOVE(I,J,Z,BIT0);
       END;
     IF (I¬=TERMINAL) THEN IF (GRAPH(I,0)=0) THEN IF (GRAPH(0,I)>0) THEN
       DO J = 1 TO N;
          IF (GRAPH(J,I)>0) THEN CALL REMOVE(J,I,Z,BIT0);
       END;
   END;
   DO I = 1 TO N;
     DO J = 1 TO N;
       IF (GRAPH(I,J) > 0) THEN IF (USED(GRAPH(I,J))) THEN
         CALL REMOVE (I,J,Z,BIT0);
     END;
   END;
   CALL P_GRAPH;
   IF (P_FLAG) THEN CALL RULE_THREE;
   RETURN;
 END RULE_TWO;
 END RULE_ONE;
1/****************************************************************/
 /*   RULE THREE FINDS ALL CHILDREN OF THE GIVEN GRAPH BY DELETING */
 /* ONE SEQUENCE AT A TIME AND CALLING P_GRAPH UNTIL ALL SEQUENCES */
 /* HAVE BEEN DELETED.  THESE CHILDREN ARE STORED IN 'CHILDREN'    */
 /* RULE FOUR IS CALLED.                                           */
 /****************************************************************/

 RULE_THREE: PROC;
   DCL (CHILDREN(M),TEMP,I,J,K) FIXED;
   DCL ANCESTOR(M) BIT(1);
   CHILDREN = 0;
   DO I = 1 TO N;
     DO J = 1 TO N;
         TEMP = GRAPH(I,J);
       IF (TEMP>0) THEN IF (WEIGHT(TEMP)=BIT0)
         THEN IF (CHILDREN(TEMP)=0) THEN DO;
         CALL REMOVE(I,J,Z,BIT0);
         CALL P_GRAPH;
         IF (P_FLAG) THEN DO;
           CHILDREN(TEMP) = I;
           K=0;
           DO WHILE(D_SEQ(D_NUM-K,1) ¬= I);
             CHILDREN(-GRAPH(D_SEQ(D_NUM-K,1),D_SEQ(D_NUM-K,2))) = -1;
             K = K + 1;
           END;
         END;
         CALL RESTORE(I,J);
       END;
     END;
   END;
   CALL RELIABILITY;
   ANCESTOR = WEIGHT;
   CALL RULE_FOUR(CHILDREN);
   WEIGHT = ANCESTOR;
   RETURN;
 END RULE_THREE;
1/*****************************************************************/
 /*   RULE FOUR DELETES THE LOWEST NUMBERED CHILD FROM CHILDREN,    */
 /* DELETES THAT CHILD FROM 'CHILDREN' AND CALLS ITSELF AGAIN. THE */
 /* RULE CONTINUES UNTIL A NON-P_GRAPH IS ENCOUNTERED AT WHICH TIME*/
 /* IT RETURNS.  IT RESTORES THE FIRST CHILD AND DELETES THE SECOND*/
 /* CHILD THEN CALLS ITSELF AGAIN.  IT CONTINUES UNTIL 'CHILDREN'  */
 /* IS EMPTY.                                                       */
 /*****************************************************************/

 RULE_FOUR: PROC (FATHERS) RECURSIVE;
   DCL (I,J,K,L,ELDER_BRO,CHILDREN(M)) FIXED INIT(1);
   DCL ANCESTOR(M) BIT(1);
   DCL FATHERS(*) FIXED;
   DO K = 1 TO M;
     CHILDREN(K) = FATHERS(K);
   END;
   L = D_NUM + 1;
   DO WHILE (I < M);
     DO WHILE (CHILDREN(I) <= 0);
       I = I + 1;
       IF (I > M) THEN RETURN;
     END;
     K = 1;
     DO WHILE (ABS(GRAPH(CHILDREN(I),K)) ¬= I);
       K = K + 1;
     END;
     ELDER_BRO = CHILDREN(I);
```

```
      J = D_NUM + 1;
      CALL REMOVE(ELDER_BRO,K,Z,BIT1);
      DO H = J TO D_NUM;
        CHILDREN(-GRAPH(D_SEQ(H,1),D_SEQ(H,2))) = 0;
      END;
      CALL P_GRAPH;
      IF (P_FLAG) THEN DO;
        CALL RELIABILITY;
        ANCESTOR = WEIGHT;
        CALL RULE_FOUR(CHILDREN);
        WEIGHT = ANCESTOR;
      END;
      ELSE DO;
        L = 0;
        DO WHILE (GRAPH(D_SEQ(D_NUM-L,1),D_SEQ(D_NUM-L,2))¬=
                  GRAPH(ELDER_BRO,K));
          WEIGHT(-GRAPH(D_SEQ(D_NUM,1),D_SEQ(D_NUM,2))) = BIT0;
          L = L + 1;
        END;
        WEIGHT(-GRAPH(ELDER_BRO,K)) = BIT0;
      END;
      CALL RESTORE(ELDER_BRO,K);
    END;
    RETURN;

  END RULE_FOUR;

1/*********************************************************************/
  /*   REMOVE IS A SUBROUTINE WHICH REMOVES A REQUESTED EDGE AND ALL   */
  /* EDGES WHICH MAY BE MEMBERS OF A SEQUENCE WHICH INCLUDES THE       */
  /* REQUESTED EDGE.  THE FOUR PARAMETERS PASSED TO THE SUBROUTINE     */
  /* ARE THE TWO VERTICIES OF THE REQUESTED EDGE, THE DIRECTION, AND   */
  /* THE WEIGHT_FLAG.                                                  */
  /*                                                                   */
  /*   THE DIRECTION IS USED SINCE THE PROCEDURE IS RECURSIVE.         */
  /* WHEN THE PROCEDURE IS INITIALLY CALLED, THE DIRECTION IS ZERO.    */
  /* THE PROCEDURE WILL THEN CHECK TO SEE IF THE BACKWARD DIRECTION    */
  /* HAS AN EDGE WHICH WILL BE A PART OF THE SEQUENCE.  IF AN EDGE     */
  /* EXISTS IN THE BACKWARD DIRECTION, THE PROCEDURE CALLS ITSELF      */
  /* WITH A DIRECTION OF -1.                                           */
  /*   THE PROCEDURE NEXT CHECKS IN THE FORWARD DIRECTION (1).         */
  /*   THE PROCEDURE THEN REMOVES THE REQUESTED EDGE.                  */
  /*   THE WEIGHT_FLAG IS USED TO DETERMINE WHETHER OR NOT TO INCLUDE */
  /* THE REMOVED EDGE IN THE WEIGHT.  A VALUE OF 1 WILL WEIGHT THE     */
  /* EDGE.                                                             */
  /*********************************************************************/

  REMOVE: PROC(OUT,IN,DIR,WEIGHT_FLAG) RECURSIVE;
    DCL (OUT,IN,DIR,I,J) FIXED;
    DCL WEIGHT_FLAG BIT(*);
  IF (T_FLAG) THEN PUT SKIP EDIT ('REMOVE')(A);
  IF (T_FLAG) THEN PUT DATA (GRAPH(OUT,IN));
    IF (GRAPH(OUT,IN) <= 0) THEN RETURN;
    J = GRAPH(OUT,IN);
```

```
   IF (DIR = 0) THEN IF (WEIGHT(J)=BIT0) THEN DO;
       D_NUM = D_NUM + 1;
       D_SEQ(D_NUM,1) = OUT;
       D_SEQ(D_NUM,2) = IN;
       IF (WEIGHT_FLAG) THEN WEIGHT(J) = BIT1;
       GRAPH(OUT,IN) = -J;
       GRAPH(OUT,0)=GRAPH(OUT,0) - 1;
       GRAPH(0,IN)=GRAPH(0,IN) - 1;
     END;
   IF (DIR < 1) THEN IF (GRAPH(OUT,0)=0) THEN IF (GRAPH(0,OUT)=1) THEN
     DO I = 1 TO N;
       IF (GRAPH(I,OUT)>0) THEN IF (WEIGHT(GRAPH(I,OUT))=BIT0) THEN DO;
         GRAPH(I,OUT)=-GRAPH(I,OUT);
         D_NUM = D_NUM + 1;
         D_SEQ(D_NUM,1) = I;
         D_SEQ(D_NUM,2) = OUT;
         IF (WEIGHT_FLAG) THEN WEIGHT(-GRAPH(I,OUT)) = BIT1;
         GRAPH(0,OUT)=0;
         GRAPH(I,0)=GRAPH(I,0) - 1;
         CALL REMOVE(I,OUT,MO,WEIGHT_FLAG);
         I = N + 1;
       END;
     END;

   IF (DIR > -1) THEN IF (GRAPH(0,IN)=0) THEN IF (GRAPH(IN,0)=1) THEN
     DO I = 1 TO N;
       IF (GRAPH(IN,I)>0) THEN IF (WEIGHT(GRAPH(IN,I)) = BIT0) THEN DO;
         GRAPH(IN,I)=-GRAPH(IN,I);
         D_NUM = D_NUM + 1;
         D_SEQ(D_NUM,1) = IN;
         D_SEQ(D_NUM,2) = I;
         IF (WEIGHT_FLAG) THEN WEIGHT(-GRAPH(IN,I)) = BIT1;
         GRAPH(IN,0)=0;
         GRAPH(0,I)=GRAPH(0,I) - 1;
         CALL REMOVE(IN,I,O,WEIGHT_FLAG);
         I = N + 1;
       END;
     END;

   RETURN;
 END REMOVE;
1/*********************************************************************/
 /*   RESTORE IS A SUBROUTINE WHICH USES THE ARRAYS D_SEQ AND GRAPH   */
 /* TO RESTORE THE REQUESTED EDGES TO THE GRAPH.  THE INFORMATION     */
 /* PASSED TO RESTORE IS THE FROM AND TO VERTICIES OF THE EDGE        */
 /* REQUESTED.  RESTORE WILL THEN RESTORE THE REQUESTED EDGE AND ALL  */
 /* EDGES REMOVE SINCE THE DELETION OF THE REQUESTED EDGE.  IN THIS   */
 /* WAY, RESTORE WILL NOT ONLY RESTORE A SINGLE EDGE BUT SEQUENCES    */
 /* WHICH MAY HAVE BEEN REMOVED INCLUDING THAT EDGE.                  */
 /*********************************************************************/
 RESTORE: PROC (OUT,IN) RECURSIVE;
   DCL (OUT,IN) FIXED;
 IF (T_FLAG) THEN PUT SKIP EDIT ('RESTORE')(A);
 IF (T_FLAG) THEN PUT DATA (GRAPH(D_SEQ(D_NUM,1),D_SEQ(D_NUM,2)));
```

```
     GRAPH(D_SEQ(D_NUM,1),D_SEQ(D_NUM,2))=
                          -GRAPH(D_SEQ(D_NUM,1),D_SEQ(D_NUM,2));
     GRAPH(D_SEQ(D_NUM,1),0)=GRAPH(D_SEQ(D_NUM,1),0)+1;
     GRAPH(0,D_SEQ(D_NUM,2))=GRAPH(0,D_SEQ(D_NUM,2))+1;
     D_NUM=D_NUM-1;
   IF (GRAPH(D_SEQ(D_NUM+1,1),D_SEQ(D_NUM+1,2))¬=GRAPH(OUT,IN))
     THEN CALL RESTORE (OUT,IN);
   RETURN;
 END RESTORE;
1/**********************************************************************/
 /*   P_GRAPH CHECKS TO SEE IF THE CURRENT GRAPH IS A P_GRAPH.  IF    */
 /* THE GRAPH IS NON-CYCLIC, THEN THE GRAPH WILL BE P_ACYCLIC.        */
 /* THE PROCEEDURE CHECKS BY SEEING IF ALL INTERNAL VERTX HAVE IN     */
 /* DEGREE AND OUT DEGREE GREATER THAN ZERO.  SINCE THE IN DEGREES    */
 /* ARE LISTED IN THE ZERO ROW AND THE OUT DEGREES ARE LISTED IN THE  */
 /* ZERO COLUMN, IT IS ONLY NECESSARY TO CHECK TO SEE IF ALL VERTICES,*/
 /* EXCLUDING THE START AND TERMINAL, HAVE NON-ZERO ENTRIES.  IF A    */
 /* NODE HAS BEEN DELETED, IT WILL HAVE BOTH IN DEGREE AND OUT DEGREE */
 /* OF ZERO.  A P_GRAPH GIVES P_FLAG = 1, ELSE P_FLAG = 0.            */
 /*********************************************************************/

 P_GRAPH: PROC;
 DCL I FIXED;
   P_FLAG = BIT1;
   IF (GRAPH(START,0) = 0) THEN P_FLAG = BIT0;
   IF (GRAPH(0,TERMINAL) = 0) THEN P_FLAG = BIT0;
   I = 0;
   DO WHILE (P_FLAG & I < N);
     I = I + 1;
     IF (I ¬= START) THEN IF (I ¬= TERMINAL)
       THEN IF (GRAPH(0,I) = 0 | GRAPH(I,0) = 0)
         THEN IF (GRAPH(0,I) ¬= GRAPH(I,0)) THEN P_FLAG = BIT0;
   END;
 IF (T_FLAG) THEN PUT DATA (P_FLAG);
   RETURN;
 END P_GRAPH;
1/*********************************************************************/
 /*     RELIABILITY TAKES THE ORIGINAL GRAPH RELIABILITY AND         */
 /* DIVIDES BY THE RELIABILITIES OF THE EDGES WHICH HAVE BEEN        */
 /* REMOVED.  THESE EDGES ARE IN THE ARRAY 'D_SEQ' AND THE NUMBER OF */
 /* EDGES WHICH HAVE BEEN REMOVED IS 'D_NUM'.  D_NUM IS A POINTER    */
 /* WHICH GIVES THE CURRENT POSITION IN 'D_SEQ'.                     */
 /********************************************************************/
 RELIABILITY: PROC;
   DCL (EDGES(M),I,J,M_NUM,N_NUM) FIXED INIT(0);
   DCL SUBGRAPH_REL FLOAT;
   EDGES = 0;
   SUBGRAPH_REL = REL;
   DO I = 1 TO D_NUM;
     J = -GRAPH(D_SEQ(I,1),D_SEQ(I,2));
     SUBGRAPH_REL=SUBGRAPH_REL/EDGE_REL(-GRAPH(D_SEQ(I,1),D_SEQ(I,2)));
     EDGES(J) = 1;
   END;
   DO I = 1 TO N;
```

```
      IF (GRAPH(0,I)>0) THEN DO;
        N_NUM = N_NUM + 1;
        M_NUM = M_NUM + GRAPH(0,I);
      END;
    END;
    SIGN = (-1) ** (M_NUM - N_NUM);
    SUBGRAPH_REL = SUBGRAPH_REL * SIGN;
    IF (I_FLAG) THEN DO I = 1 TO M;
      IF (EDGES(I) = 0) THEN
        IMPORTANCE(I)=IMPORTANCE(I)+SUBGRAPH_REL/EDGE_REL(I);
    END;
    GRAPH_REL = GRAPH_REL + SUBGRAPH_REL;
    IF (L_FLAG) THEN DO;
      PUT SKIP(2);
      PUT EDIT (SIGN,'')(F(8),COL(20),A);
      DO I = 1 TO M;
        IF (EDGES(I) = 0) THEN PUT EDIT (I)(F(3));
      END;
      PUT EDIT (SUBGRAPH_REL)(COL (60),F(8,4));
      LINE = LINE + 2;
      IF (LINE > 60) THEN CALL LINKSET;
    END;
    RETURN;
  END RELIABILITY;
      END;
1/*****************************************************************/
 /*     DATA_IN INPUTS THE ORIGINAL DATA, ECHO PRINTS IT, INPUTS  */
 /* THE DATA INTO 'GRAPH' AND PRINTS THE ARRAY 'GRAPH'.           */
 /*****************************************************************/

 DATA_IN: PROC;
   DCL (DATA(2*M,3),I,J,K) FIXED;
   DCL REL FLOAT;
   IF (D_FLAG) THEN DO;
     PUT SKIP(4) EDIT ('INPUT DATA')(COL (1),A);

     PUT SKIP EDIT('     EDGE NO. FROM    TO  RELIABILITY')(A);
   END;
   K = 0;
   DO I = 1 TO M;
     K = K + 1;
     GET FILE(INPUT) LIST(DATA(K,1),DATA(K,2),DATA(K,3),REL);
     J = GRAPH(DATA(K,1),DATA(K,2));
     IF (J ¬= 0) THEN DO;
       EDGE_REL(J) = EDGE_REL(J) + REL - EDGE_REL(J) * REL;
       K = K - 1;
       M = M - 1;
     END;
     ELSE DO;
       GRAPH(DATA(K,1),DATA(K,2)) = K;
       EDGE_REL(K) = REL;
     END;
     IF (DATA(K,3) = 0) THEN
       IF (GRAPH(DATA(K,2),DATA(K,1)) ¬= 0) THEN DO;
```

```
          J = GRAPH(DATA(K,2),DATA(K,1));
          EDGE_REL(J) = EDGE_REL(J) + REL - EDGE_REL(J) * REL;
        END;
        ELSE DO;
          M = M + 1;
          GRAPH(DATA(K,2),DATA(K,1)) = M;
          EDGE_REL(M) = REL;
          DATA(M,2)=DATA(K,1);
          DATA(M,1)=DATA(K,2);
          DATA(M,3)=DATA(K,3);
        END;
      END;

  IF (D_FLAG) THEN DO;
    PUT SKIP(4) EDIT ('')(COL (40),A);
    IF (M_FLAG) THEN DO I = 1 TO N;
      PUT EDIT ('  ____')(A);
    END;
    DO I = 1 TO N;
      LINE = LINE + 3;
      IF LINE > 60 THEN CALL HEADER;
      PUT EDIT ('')(COL (40),A);
      IF (M_FLAG) THEN DO J = 1 TO N+1;
        PUT EDIT ('|     ')(A);
      END;
      PUT SKIP EDIT(I,DATA(I,1),DATA(I,2),EDGE_REL(I))
                      ((3)F(7),F(8,2));
      IF (M_FLAG) THEN DO;
        PUT EDIT('')(COL (40),A);
        DO J = 1 TO N;
          PUT EDIT ('| ',GRAPH(I,J),' ')(A,F(2),A);
        END;
        PUT EDIT ('|','')(A,COL (40),A);
        DO J = 1 TO N;
          PUT EDIT ('|____')(A);
        END;
        PUT EDIT ('|')(A);
      END;
    END;
     PUT SKIP;
     IF (M > N) THEN
    DO I = N+1 TO M;
      LINE = LINE + 3;
      IF LINE > 60 THEN CALL HEADER;
      PUT SKIP EDIT(I,DATA(I,1),DATA(I,2),EDGE_REL(I))
                   ((3)F(7),F(8,2));
      PUT SKIP(2);
    END;
    END;
    RETURN;
  END DATA_IN;
1/*******************************************************************/
 /*   LINKSET IS CALLED DURING THE PRINTING OF THE LINKSET.  IT    */
 /* PRINTS A HEADING FOR THE OUTPUT.                               */
 /*******************************************************************/
 LINKSET: PROC;
   CALL HEADER;
   PUT EDIT('LINKSET')(A);
   PUT SKIP EDIT ('SIGN      SUBGRAPH                     RELIABILITY')
                 (A);
   PUT SKIP(4);
   LINE = 18;
 RETURN;
 END LINKSET;
 END;
1/*******************************************************************/
 /*  HEADER PRINTS A HEADING AT THE TOP OF EACH PAGE.             */
 /*******************************************************************/
 HEADER: PROC;
   PUT PAGE;
   PUT SKIP EDIT('STRUCTURED TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM')
                (COL (1),A);
   PUT SKIP(4);
   LINE = 15;
   RETURN;
 END HEADER;
 END MAIN;
//
```

Appendix B

Minimum Paths (MPM)

```
//B14805A JOB (12817,000-00-0000),CLASS=A,TIME=(,40),
// MSGCLASS=A,NOTIFY=*
/*PASSWORD ?
/*ROUTE PRINT LOCAL
/*JOBPARM ROOM=B,FORMS=9021,COPIES=3
// EXEC PLC
//INPUT DD DSN=U14805A.SAMPLE.DATA(C),DISP=SHR
//SYSIN DD *
*PL/C TIME=(,40) PAGES=500
 MAIN: PROC OPTIONS(MAIN);
   DCL (N,M,START,TERMINAL) FIXED;
   DCL COMMENTS CHAR(80) VAR;
   DCL (D_FLAG,C_FLAG) BIT(1) INIT('1');
   ON ENDFILE(INPUT) C_FLAG = '0';
   GET FILE(INPUT) EDIT (COMMENTS) (COL(1),A(80));
     C_FLAG = '1';
   DO WHILE (C_FLAG);
     IF LENGTH(COMMENTS) > 3 THEN IF SUBSTR(COMMENTS,1,4) ¬= '$JOB'
     THEN DO;
       PUT SKIP EDIT (COMMENTS) (A);
       GET FILE(INPUT) EDIT (COMMENTS) (COL(1),A(80));
     END;
     ELSE C_FLAG = '0';
   END;
   GET FILE(INPUT) LIST(N,M,START,TERMINAL);

   BEGIN;
     DCL (PATH(2*M),UNA(N),PATHS(2*M,2*M),GRAPH(N,N),N1) FIXED;
     DCL (REL,EDGE_REL(2*M),GRAPH_REL) FLOAT;
     EDGE_REL = 0.0;
     GRAPH_REL = 0;
     GRAPH = 0;
     N1 = 0;
     UNA = 0;
     PATHS = 0;
     PATH = 0;
     CALL DATA_IN;
     PUT EDIT ('MINIMUM PATHS') (A);
     CALL SEARCH (START);
     CALL GRAPHS;
     PUT SKIP EDIT ('RELIABILITY =',-GRAPH_REL)
        (COL(20),A,F(6,4));

 SEARCH: PROC(START) RECURSIVE;
   DCL (START,I,L,K) FIXED;
     IF (START = TERMINAL) THEN GOTO MIN_PATHS;
     UNA(START) = 1;
     DO I = 1 TO N;
       IF (UNA(I) = 0) THEN IF (GRAPH(START,I) > 0) THEN DO;
         PATH(GRAPH(START,I)) = 1;
         CALL SEARCH(I);
         PATH(GRAPH(START,I)) = 0;
         UNA(I) = 0;
       END;
     END;
     RETURN;

MIN_PATHS:;
   N1 = N1 + 1;
   PUT SKIP EDIT (N1) (F(5),COL (20));
   DO L = 1 TO M;
     IF PATH(L) = 1 THEN DO;
       PATHS(N1,L) = 1;
       PUT EDIT (L)(F(4));
     END;
   END;
   RETURN;
END SEARCH;
GRAPHS: PROC;
   DCL (SERIES(N1),N2) FIXED;
     N2 = 1;
     SERIES = 0;
     SERIES(1) = 1;
     DO WHILE (SERIES(1) ¬= N1);
       CALL RELIABILITY;
       IF SERIES(N2) = N1 THEN DO;
         SERIES(N2) = 0;
         SERIES(N2-1) = SERIES(N2-1) + 1;
         N2 = N2 - 1;
       END;
       ELSE DO;
         N2 = N2 + 1;
         SERIES(N2) = SERIES(N2-1) + 1;
       END;
     END;
     CALL RELIABILITY;
     RETURN;
RELIABILITY: PROC;
   DCL (I,COMBO(M)) FIXED;
   REL = 1;
   COMBO = 0;
   DO I = 1 TO N2;
     DO J = 1 TO M;
       IF PATHS(SERIES(I),J) = 1 THEN COMBO (J) = 1;
     END;
   END;
   DO I = 1 TO M;
     IF COMBO(I) = 1 THEN DO;
       REL = REL * EDGE_REL(I);
     END;
   END;
   GRAPH_REL =  -(GRAPH_REL + REL);
RETURN;
END RELIABILITY;
END GRAPHS;
DATA_IN: PROC;
   DCL (DATA(2*M,3),I,J,K) FIXED;
   PUT SKIP(4) EDIT ('INPUT DATA')(COL (1),A);
```

```
K = 0;
 PUT SKIP EDIT('     EDGE NO. FROM    TO  RELIABILITY')(A);
DO I = 1 TO M;
  K = K + 1;
  GET FILE(INPUT) LIST(DATA(K,1),DATA(K,2),DATA(K,3),REL);
  IF (GRAPH(DATA(K,1),DATA(K,2)) ¬= 0) THEN DO;
    J = GRAPH(DATA(K,1),DATA(K,2));
    EDGE_REL(J) = EDGE_REL(J) + REL - EDGE_REL(J) * REL;
    K = K - 1;
    M = M - 1;
  END;
  ELSE DO;
    GRAPH(DATA(K,1),DATA(K,2)) = K;
    EDGE_REL(K) = FLOAT(REL);
  END;
  IF (DATA(K,3) = 0) THEN
    IF (GRAPH(DATA(K,2),DATA(K,1)) ¬= 0) THEN DO;
      J = GRAPH(DATA(K,2),DATA(K,1));
      EDGE_REL(J) = EDGE_REL(J) + REL - EDGE_REL(J) * REL;
    END;
    ELSE DO;
      M = M + 1;
      GRAPH(DATA(K,2),DATA(K,1)) = M;
      EDGE_REL(M) = REL;
      DATA(M,2)=DATA(K,1);
      DATA(M,1)=DATA(K,2);
      DATA(M,3)=DATA(K,3);
    END;
  END;

PUT SKIP(4) EDIT ('')(COL (40),A);
DO I = 1 TO N;
  PUT EDIT (' ____')(A);
END;
DO I = 1 TO N;
  PUT EDIT ('')(COL (40),A);
  DO J = 1 TO N+1;
    PUT EDIT ('|    ')(A);
  END;
  PUT SKIP EDIT(I,DATA(I,1),DATA(I,2),EDGE_REL(I))
                 ((3)F(7),F(8.2));
  PUT EDIT('')(COL (40),A);
  DO J = 1 TO N;
    PUT EDIT ('| ',GRAPH(I,J),' ')(A,F(2),A);
  END;
  PUT EDIT ('|','')(A,COL (40),A);
  DO J = 1 TO N;
    PUT EDIT ('|____')(A);
  END;
  PUT EDIT ('|')(A);
END;
 PUT SKIP;
 IF (M > N) THEN
```

```
   DO I = N+1 TO M;
     PUT SKIP EDIT(I,DATA(I,1),DATA(I,2),EDGE_REL(I))
                 ((3)F(7),F(8.2));
     PUT SKIP(2);
   END;
   RETURN;
 END DATA_IN;
 END MAIN;
//
```

Appendix C

Sample Input Data

```
EXAMPLE THREE
FROM 'NEW TOPOLOGICAL FORMULA  SATYANARAYANA & PRABHAKAR
IEEE TRANS. ON RELIABILITY  PAGE 85  ARPA NETWORK.
A COMPLEX SOURCE-TO-TERMINAL RELIABILITY NETWORK  CONTAINING
6 CYCLES.
$JOB MATRIX LINKSET IMPORTANCE
6 12 1 6
1 2 1 .9
1 3 1 .9
4 6 1 .9
2 4 1 .9
5 6 1 .9
3 5 1 .9
4 5 1 .9
2 5 1 .9
2 3 1 .9
5 4 1 .9
5 2 1 .9
3 2 1 .9
```

Appendix D

Sample Output

REFERENCES

[1] A. Satyanarayana, A. Prabhaker, "New topological formula and rapid algorithm for reliability analysis of complex networks," IEEE Trans. Reliability, vol. R-27, 1978 Jun, pp. 82-100.

[2] R.R. Willie, "A theorem concerning cyclic directed graphs with applications to network reliability," Networks, vol. 10, 1980, pp. 71-78.

[3] A. Satyanarayana, J.N. Hagstrom, "Combinatorial properties of directed graph useful in network reliability," Networks, vol. 11, 1981, pp. 357-366.

[4] A. Satyanarayana, "A unified formula for analysis of some network reliability problems," IEEE Trans. Reliability vol. R-31, 1982 Apr, pp. 23-32.

[5] M.O. Locks, "System reliability analysis: A tutorial," Microelectron Reliability, vol. 18, 1978, pp. 335-345.

[6] C.C. Bolaki, Structured Topological Reliability Analysis Program, MBA Research Report, Department of Administrative Sciences, Oklahoma State University (1983).

[7] R. Tarjan, "Depth-first-search, linear graph algorithms," Siam J. Comput., vol. 1, 1972, pp. 146-160.

[8] J.L. Burris, Model for the analysis of probability of systems, MBA Research Report, Department of Administrative Sciences, Oklahoma State University (1976).

```
                                       J E S 2   J O B   L O G

19.50.36 JOB 2435  OSU517I                                                                   JOB READ ON     CLASS A     B14805A
19.50.37 JOB 2435  $HASP373 B14805A  STARTED - INIT  4 - CLASS A - SYS 3081
19.50.37 JOB 2435  OSU518I                                                                   JOB START       CLASS A     B14805A
19.50.55 JOB 2435  OSU519I                                                                   JOB END         CLASS A     B14805A
19.50.55 JOB 2435  $HASP395 B14805A  ENDED

------ JES2 JOB STATISTICS ------

 03 MAY 84 JOB EXECUTION DATE

       172 CARDS READ

       307 SYSOUT PRINT RECORDS

         0 SYSOUT PUNCH RECORDS

      0.29 MINUTES EXECUTION TIME
```

```
    1        //B14805A JOB (XXXXX,                                                               JOB 2435
             // 000-00-0000),CLASS=A,TIME=(,40),
             // MSGCLASS=A,
             // NOTIFY=U14805A
             ***ROUTE PRINT LOCAL                                                                0000004
             ***JOBPARM ROOM=B,FORMS=9021,COPIES=3                                               0000005
    2        // EXEC PLC                                                                        00000060
    8        //INPUT DD DSN=U14805A.SAMPLE.DATA(C),DISP=SHR                                      00000070
    9        //SYSIN DD *                                                                       00000080
```

```
IEF142I B14805A GO - STEP WAS EXECUTED - COND CODE 0000
IEF373I STEP /GO      / START 84124.1950
IEF374I STEP /GO      / STOP  84124.1950 CPU    0MIN 14.82SEC SRB    0MIN 00.02SEC VIRT   256K SYS   232K
IEF375I  JOB /B14805A / START 84124.1950
IEF376I  JOB /B14805A / STOP  84124.1950 CPU    0MIN 14.82SEC SRB    0MIN 00.02SEC
+---------------------------------------------------------------------+
| PROCESSOR TIME ----------0.00421 CPU HOURS @ $1,260.00 ---------5.30 |
| PROCESSOR STORAGE -------1.05387 K-BYTE HOURS @  $0.50 ---------0.53 |
|                                  TOTAL PROCESSOR COST --------$5.83 |
|                                                                     |
| DISK EXCPS --------------------1 @  $0.36 PER 1000 ------------0.00 |
|             I/O COST (EXCLUDING PRINTER/READER/PUNCH) -------$0.00 |
|                                                                     |
| TOTAL COST (AFTER       $1.75   2ND SHIFT DISCOUNT) ----------$4.08 |
| AMOUNT OF FUNDS REMAINING ---------------------------------$106.80 |
| EXCLUDING CURRENT CHARGES FOR NON-COMPUTER SERVICES                 |
+---------------------------------------------------------------------+
*** NO ID ***      05/03/84 19:50    PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   *** NO ID ***
*** NO ID ***      05/03/84 19:50    PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   *** NO ID ***
*** NO ID ***      05/03/84 19:50    PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   *** NO ID ***
*** NO ID ***      05/03/84 19:50    PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   *** NO ID ***
*** NO ID ***      05/03/84 19:50    PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   *** NO ID ***
*** NO ID ***      05/03/84 19:50    PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   *** NO ID ***
*** NO ID ***      05/03/84 19:50    PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   *** NO ID ***
*** NO ID ***      05/03/84 19:50    PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   *** NO ID ***
*** NO ID ***      05/03/84 19:50    PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   *** NO ID ***
*** NO ID ***      05/03/84 19:50    PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   PL/C   *** NO ID ***
```

```
*PL/C TIME=(,40) PAGES=500                                                  00000090

*OPTIONS IN EFFECT*    TIME=(0.40.00),PAGES=500,LINES=30000,NOATR,NOXREF,FLAGW,NOCMNTS,SORMGIN=(2,72,1),ERRORS=(10,50),
*OPTIONS IN EFFECT*    TABSIZE=4900,SOURCE,OPLIST,NOCMPRS,HDRPG,AUXIO=10000,LINECT=60,NOALIST,MONITOR=(UDEF,NOBNDRY,
*OPTIONS IN EFFECT*    SUBRG,AUTO),MCALL,NOMTEXT,DUMP=(S,F,L,E,U,R),DUMPE=(S,F,L,E,U,R),DUMPT=(S,F,L,E,U,R)

 MAIN: PROC OPTIONS(MAIN);                                                  00000100   PL/C-R7.6-041 05/03/84 19:50 PAGE   1

  STMT LEVEL NEST BLOCK MLVL  SOURCE TEXT

     1                        MAIN: PROC OPTIONS(MAIN);                                               00000100
     2     1           1        DCL (N,M,START,TERMINAL) FIXED;                                       00000110
     3     1           1        DCL COMMENTS CHAR(80) VAR;                                            00000120
     4     1           1        DCL (D_FLAG,C_FLAG) BIT(1) INIT('1');                                 00000130
     5     1           1        ON ENDFILE(INPUT) C_FLAG = '0';                                       00000140
     7     2           2        GET FILE(INPUT) EDIT (COMMENTS) (COL(1),A(80));                       00000150
     8     1           1          C_FLAG = '1';                                                       00000160
     9     1           1        DO WHILE (C_FLAG);                                                    00000170
    10     1    1      1          IF LENGTH(COMMENTS) > 3 THEN IF SUBSTR(COMMENTS,1,4) ¬= '$JOB'      00000180
    12     1    1      1          THEN DO;                                                            00000190
    13     1    2      1            PUT SKIP EDIT (COMMENTS) (A);                                     00000200
    14     1    2      1            GET FILE(INPUT) EDIT (COMMENTS) (COL(1),A(80));                   00000210
    15     1    2      1          END;                                                                00000220
    16     1    1      1          ELSE C_FLAG = '0';                                                  00000230
    17     1    1      1        END;                                                                  00000240
    18     1           1        GET FILE(INPUT) LIST(N,M,START,TERMINAL);                             00000250
                                                                                                      00000260
    19     1           1        BEGIN;                                                                00000270
    20     2           3          DCL (PATH(2*M),UNA(N),PATHS(2*M,2*M),GRAPH(N,N),N1) FIXED;          00000280
    21     2           3          DCL (REL,EDGE_REL(2*M),GRAPH_REL) FLOAT;                            00000290
    22     2           3          EDGE_REL = 0.0;                                                     00000300
    23     2           3          GRAPH_REL = 0;                                                      00000310
    24     2           3          GRAPH = 0;                                                          00000320
    25     2           3          N1 = 0;                                                             00000330
    26     2           3          UNA = 0;                                                            00000340
    27     2           3          PATHS = 0;                                                          00000350
    28     2           3          PATH = 0;                                                           00000360
    29     2           3          CALL DATA_IN;                                                       00000370
    30     2           3          PUT EDIT ('MINIMUM PATHS') (A);                                     00000380
    31     2           3          CALL SEARCH (START);                                                00000390
    32     2           3          CALL GRAPHS;                                                        00000400
    33     2           3          PUT SKIP EDIT ('RELIABILITY =',-GRAPH_REL)                          00000410
                                      (COL(20),A,F(6,4));                                             00000420
                                                                                                      00000430
    34     2           3      SEARCH: PROC(START) RECURSIVE;                                          00000440
    35     3           4        DCL (START,I,L,K) FIXED;                                              00000450
    36     3           4          IF (START = TERMINAL) THEN GOTO MIN_PATHS;                          00000460
    38     3           4          UNA(START) = 1;                                                     00000500
    39     3           4          DO I = 1 TO N;                                                      00000510
    40     3    1      4            IF (UNA(I) = 0) THEN IF (GRAPH(START,I) > 0) THEN DO;             00000530
    43     3    2      4              PATH(GRAPH(START,I)) = 1;                                       00000540
    44     3    2      4              CALL SEARCH(I);                                                 00000550
    45     3    2      4              PATH(GRAPH(START,I)) = 0;                                       00000560
    46     3    2      4              UNA(I) = 0;                                                     00000570
    47     3    2      4            END;                                                              00000580
    48     3    1      4          END;                                                                00000590
    49     3           4          RETURN;                                                             00000600
```

```
MAIN: PROC OPTIONS(MAIN);                                        00000100   PL/C-R7.6-041 05/03/84 19:50 PAGE   2
  STMT LEVEL NEST BLOCK MLVL  SOURCE TEXT

                                                                                       00000610
   50    3          4         MIN_PATHS:;                                              00000620
   51    3          4           N1 = N1 + 1;                                           00000630
   52    3          4           PUT SKIP EDIT (N1) (F(5),COL (20));                    00000640
   53    3          4           DO L = 1 TO M;                                         00000650
   54    3     1    4             IF PATH(L) = 1 THEN DO;                              00000660
   56    3     2    4               PATHS(N1,L) = 1;                                   00000670
   57    3     2    4               PUT EDIT (L)(F(4));                                00000680
   58    3     2    4             END;                                                 00000690
   59    3     1    4           END;                                                   00000700
   60    3          4           RETURN;                                                00000710
   61    3          4         END SEARCH;                                              00000730
   62    2          3         GRAPHS: PROC;                                            00000740
   63    3          5           DCL (SERIES(N1),N2) FIXED;                             00000750
   64    3          5             N2 = 1;                                              00000770
   65    3          5             SERIES = 0;                                          00000780
   66    3          5             SERIES(1) = 1;                                       00000790
   67    3          5             DO WHILE (SERIES(1) ¬= N1);                          00000800
   68    3     1    5               CALL RELIABILITY;                                  00000810
   69    3     1    5               IF SERIES(N2) = N1 THEN DO;                        00000820
   71    3     2    5                 SERIES(N2) = 0;                                  00000830
   72    3     2    5                 SERIES(N2-1) = SERIES(N2-1) + 1;                 00000840
   73    3     2    5                 N2 = N2 - 1;                                     00000850
   74    3     2    5               END;                                               00000860
   75    3     1    5               ELSE DO;                                           00000870
   76    3     2    5                 N2 = N2 + 1;                                     00000880
   77    3     2    5                 SERIES(N2) = SERIES(N2-1) + 1;                   00000890
   78    3     2    5               END;                                               00000900
   79    3     1    5             END;                                                 00000910
   80    3          5             CALL RELIABILITY;                                    00000920
   81    3          5             RETURN;                                              00000930
   82    3          5         RELIABILITY: PROC;                                       00000940
   83    4          6           DCL (I,COMBO(M)) FIXED;                                00000950
   84    4          6           REL = 1;                                               00000960
   85    4          6           COMBO = 0;                                             00000970
   86    4          6           DO I = 1 TO N2;                                        00000980
   87    4     1    6             DO J = 1 TO M;                                       00000990
   88    4     2    6               IF PATHS(SERIES(I),J) = 1 THEN COMBO (J) = 1;      00001000
   90    4     2    6             END;                                                 00001010
   91    4     1    6           END;                                                   00001020
   92    4          6           DO I = 1 TO M;                                         00001040
   93    4     1    6             IF COMBO(I) = 1 THEN DO;                             00001050
   95    4     2    6               REL = REL * EDGE_REL(I);                           00001070
   96    4     2    6             END;                                                 00001080
   97    4     1    6           END;                                                   00001090
   98    4          6           GRAPH_REL =  -(GRAPH_REL + REL);                       00001100
   99    4          6         RETURN;                                                  00001110
  100    4          6         END RELIABILITY;                                         00001120
  101    3          5         END GRAPHS;                                              00001130
  102    2          3         DATA_IN: PROC;                                           00001140
  103    3          7           DCL (DATA(2*M,3),I,J,K) FIXED;                         00001150
  104    3          7           PUT SKIP(4) EDIT ('INPUT DATA')(COL (1),A);            00001160
                                                                                       00001170
  105    3          7           K = 0;                                                 00001180
  106    3          7            PUT SKIP EDIT('     EDGE NO. FROM     TO  RELIABILITY')(A);   00001190
```

```
 MAIN: PROC OPTIONS(MAIN);                                                    00000100   PL/C-R7.6-041 05/03/84 19:50 PAGE   3

  STMT LEVEL NEST BLOCK MLVL  SOURCE TEXT

   107    3          7              DO I = 1 TO M;                                                            00001200
   108    3     1    7                K = K + 1;                                                              00001210
   109    3     1    7                GET FILE(INPUT) LIST(DATA(K,1),DATA(K,2),DATA(K,3),REL);                00001220
   110    3     1    7                IF (GRAPH(DATA(K,1),DATA(K,2)) ¬= 0) THEN DO;                           00001230
   112    3     2    7                  J = GRAPH(DATA(K,1),DATA(K,2));                                       00001240
   113    3     2    7                  EDGE_REL(J) = EDGE_REL(J) + REL - EDGE_REL(J) * REL;                  00001250
   114    3     2    7                  K = K - 1;                                                            00001260
   115    3     2    7                  M = M - 1;                                                            00001270
   116    3     2    7                END;                                                                    00001280
   117    3     1    7                ELSE DO;                                                                00001290
   118    3     2    7                  GRAPH(DATA(K,1),DATA(K,2)) = K;                                       00001300
   119    3     2    7                  EDGE_REL(K) = FLOAT(REL);                                             00001310
   120    3     2    7                END;                                                                    00001320
   121    3     1    7                IF (DATA(K,3) = 0) THEN                                                 00001330
   122    3     1    7                  IF (GRAPH(DATA(K,2),DATA(K,1)) ¬= 0) THEN DO;                         00001340
   124    3     2    7                    J = GRAPH(DATA(K,2),DATA(K,1));                                     00001350
   125    3     2    7                    EDGE_REL(J) = EDGE_REL(J) + REL - EDGE_REL(J) * REL;                00001360
   126    3     2    7                  END;                                                                  00001370
   127    3     1    7                  ELSE DO;                                                              00001380
   128    3     2    7                    M = M + 1;                                                          00001390
   129    3     2    7                    GRAPH(DATA(K,2),DATA(K,1)) = M;                                     00001400
   130    3     2    7                    EDGE_REL(M) = REL;                                                  00001410
   131    3     2    7                    DATA(M,2)=DATA(K,1);                                                00001420
   132    3     2    7                    DATA(M,1)=DATA(K,2);                                                00001430
   133    3     2    7                    DATA(M,3)=DATA(K,3);                                                00001440
   134    3     2    7                  END;                                                                  00001450
   135    3     1    7                END;                                                                    00001460
                                                                                                              00001470
   136    3          7              PUT SKIP(4) EDIT ('')(COL (40),A);                                        00001480
   137    3          7              DO I = 1 TO N;                                                            00001490
   138    3     1    7                PUT EDIT (' _____')(A);                                                 00001500
   139    3     1    7              END;                                                                      00001510
   140    3          7              DO I = 1 TO N;                                                            00001520
   141    3     1    7                PUT EDIT ('')(COL (40),A);                                              00001530
   142    3     1    7                DO J = 1 TO N+1;                                                        00001540
   143    3     2    7                  PUT EDIT ('|     ')(A);                                             00001550
   144    3     2    7                END;                                                                    00001560
   145    3     1    7                PUT SKIP EDIT(I,DATA(I,1),DATA(I,2),EDGE_REL(I))                        00001570
                                                  ((3)F(7),F(8.2));                                           00001580
   146    3     1    7                PUT EDIT('')(COL (40),A);                                               00001590
   147    3     1    7                DO J = 1 TO N;                                                          00001600
   148    3     2    7                  PUT EDIT ('| ',GRAPH(I,J),' ')(A,F(2),A);                             00001610
   149    3     2    7                END;                                                                    00001620
   150    3     1    7                PUT EDIT ('|','')(A,COL (40),A);                                        00001630
   151    3     1    7                DO J = 1 TO N;                                                          00001640
   152    3     2    7                  PUT EDIT ('|_____')(A);                                               00001650
   153    3     2    7                END;                                                                    00001660
   154    3     1    7                PUT EDIT ('|')(A);                                                      00001670
   155    3     1    7              END;                                                                      00001680
   156    3          7               PUT SKIP;                                                                00001690
   157    3          7               IF (M > N) THEN                                                          00001700
   158    3          7              DO I = N+1 TO M;                                                          00001710
   159    3     1    7                PUT SKIP EDIT(I,DATA(I,1),DATA(I,2),EDGE_REL(I))                        00001720
                                                ((3)F(7),F(8.2));                                             00001730
   160    3     1    7                PUT SKIP(2);                                                            00001740
 MAIN: PROC OPTIONS(MAIN);                                              00000100   PL/C-R7.6-041 05/03/84 19:50 PAGE   4

  STMT LEVEL NEST BLOCK MLVL  SOURCE TEXT

   161    3     1     7           END;                                                           00001750
   162    3           7           RETURN;                                                        00001760
   163    3           7         END DATA_IN;                                                     00001770
   164    2           3         END MAIN;                                                        00001780

ERRORS/WARNINGS DETECTED DURING CODE GENERATION:

       WARNING: NO FILE SPECIFIED. SYSIN/SYSPRINT ASSUMED. (CGOC)
```

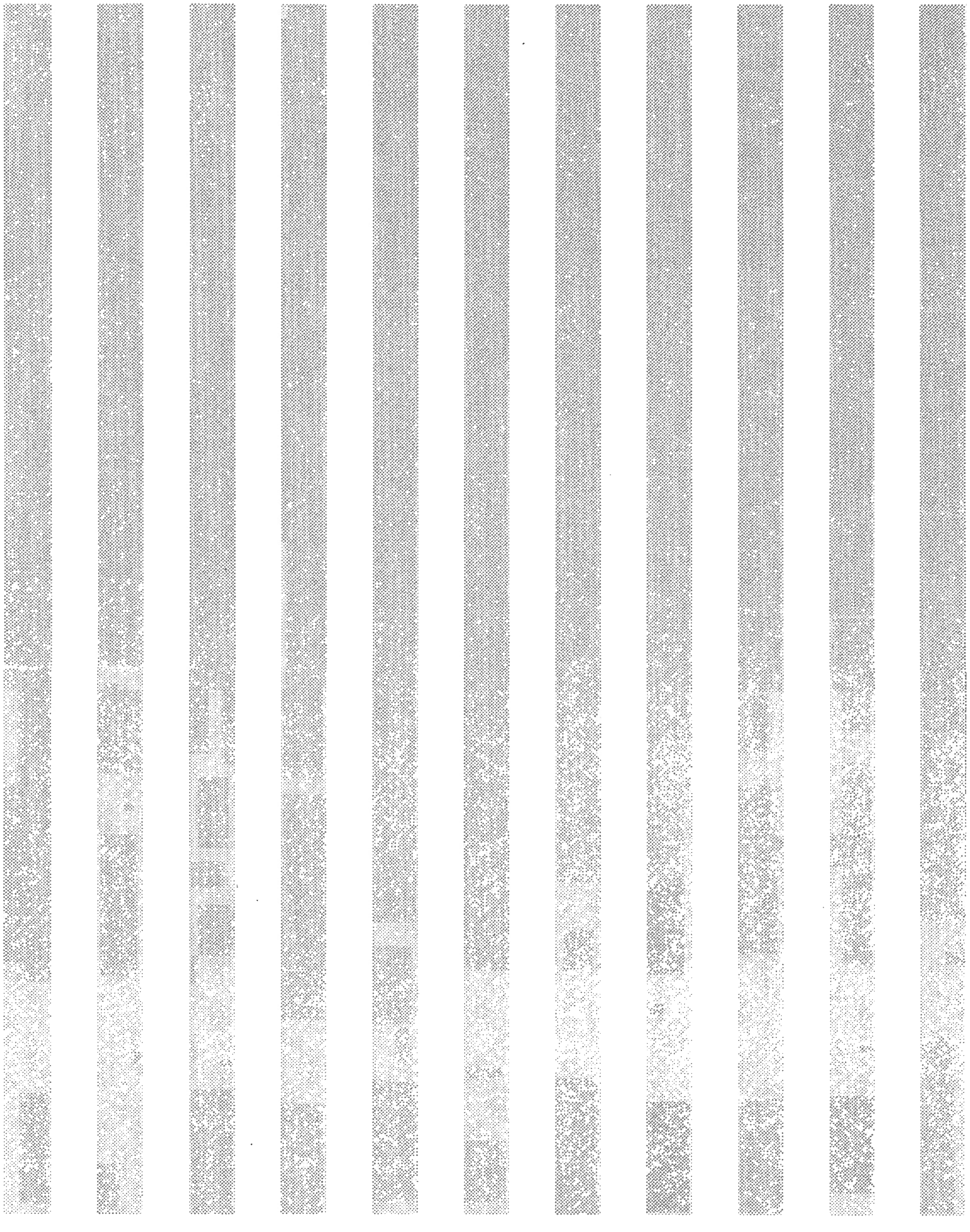
```
EXAMPLE THREE
FROM 'NEW TOPOLOGICAL FORMULA  SATYANARAYANA & PRABHAKAR
IEEE TRANS. ON RELIABILITY  PAGE 85  ARPA NETWORK.
A COMPLEX SOURCE-TO-TERMINAL RELIABILITY NETWORK  CONTAINING
6 CYCLES.

INPUT DATA
     EDGE NO. FROM    TO  RELIABILITY


       1       1       2        1           |  0 |  1 |  2 |  0 |  0 |  0 |

       2       1       3        1           |  0 |  0 |  9 |  4 |  8 |  0 |

       3       4       6        1           |  0 | 12 |  0 |  0 |  6 |  0 |

       4       2       4        1           |  0 |  0 |  0 |  0 |  7 |  3 |

       5       5       6        1           |  0 | 11 |  0 | 10 |  0 |  5 |

       6       3       5        1           |  0 |  0 |  0 |  0 |  0 |  0 |

       7       4       5        1

       8       2       5        1

       9       2       3        1

      10       5       4        1

      11       5       2        1

      12       3       2        1
MINIMUM PATHS
    1   1   3   6   9  10
    2   1   5   6   9
    3   1   4   5   7
    4   1   3   4
    5   1   3   8  10
    6   1   5   8
    7   2   4   5   7  12
    8   2   3   4  12
```

```
     9    2    3    8   10   12
    10    2    5    8   12
    11    2    3    4    6   11
    12    2    3    6   10
    13    2    5    6
                         RELIABILITY =0.9772

 IN STMT  165  PROGRAM RETURNS FROM MAIN PROCEDURE.
 MAIN: PROC OPTIONS(MAIN);                                                        00000100   PL/C-R7   POST-MORTEM DUMP    PAGE    1

IN STMT  165  SCALARS AND BLOCK-TRACE:

***** MAIN PROCEDURE MAIN
J=        13                C_FLAG='0'B             D_FLAG='1'B              COMMENTS='$JOB MATRIX LINKSET IMPORTANCE
                                                  '    TERMINAL=        6      . START=        1            M=       12
N=        6


NON-0 PROCEDURE EXECUTION COUNTS:
NAME        STMT COUNT  NAME          STMT COUNT  NAME          STMT COUNT  NAME          STMT COUNT  NAME          STMT COUNT
RELIABILITY 0082 08191  GRAPHS        0062 00001  SEARCH        0034 00032  DATA_IN       0102 00001  MAIN          0001 00001

LABEL EXECUTION COUNTS:
NAME        STMT COUNT  NAME          STMT COUNT  NAME          STMT COUNT  NAME          STMT COUNT  NAME          STMT COUNT
MIN_PATHS   0050 00013


        COMPILATION STATISTICS  (0164 STATEMENTS)  |                      EXECUTION STATISTICS
SECONDS  ERRORS  WARNINGS  PAGES  LINES  CARDS  INCL'S | SECONDS  ERRORS  WARNINGS  PAGES  LINES  CARDS  INCL'S  AUX I/O
    .11       0         1      4    193    162      0 |  14.66        0         0      2     70      0       0       19
-----------------------------------------------------+------------------------------------------------------------------
BYTES     SYMBOL TABLE    INTERMEDIATE CODE    OBJECT CODE |    STATIC CORE    AUTOMATIC CORE     DYNAMIC CORE    TOTAL STORAGE
USED       5224(   6K)        4666(    5K)     10264(  11K) |      342(    1K)       7567(    8K)     15156(  15K)     38067(  38K)
UNUSED    14376(  14K)       14762(   14K)     24334(  23K) |     1361(    1K)       1361(    1K)      1361(   1K)      1361(   1K)
AUXILIARY I/O:

  INPUT:     19 RECORDS FROM INPUT
THIS PROGRAM MAY BE RERUN WITHOUT CHANGE IN A REGION   1K BYTES SMALLER USING TABLESIZE=   1306
```

```
                                      J E S 2   J O B   L O G

19.51.36 JOB 2437  $HASP373 B14805A  STARTED - INIT  3 - CLASS F - SYS 3081
19.51.40 JOB 2437  $HASP395 B14805A  ENDED

------ JES2 JOB STATISTICS ------

 03 MAY 84 JOB EXECUTION DATE

        619 CARDS READ

        319 SYSOUT PRINT RECORDS

          0 SYSOUT PUNCH RECORDS

       0.05 MINUTES EXECUTION TIME
```

```
    1      //B14805A JOB (XXXXX,                                                    JOB 2437
           // 000-00-0000),CLASS=F,TIME=(,5),
           // MSGCLASS=A,
           // NOTIFY=U14805A
           ***ROUTE PRINT LOCAL                                                     0000004
           ***JOBPARM ROOM=B,FORMS=9021,COPIES=3                                    0000005
    2      // EXEC PLC,REGION=500K                                                 00000060
    8      //INPUT DD DSN=U14805A.SAMPLE.DATA(C),DISP=SHR                          00000070
    9      //SYSIN DD *                                                            00000080
```

```
IEF142I B14805A GO - STEP WAS EXECUTED - COND CODE 0000
IEF373I STEP /GO      / START 84124.1951
IEF374I STEP /GO      / STOP  84124.1951 CPU    0MIN 00.98SEC SRB    0MIN 00.02SEC VIRT   500K SYS   224K
IEF375I  JOB /B14805A / START 84124.1951
IEF376I  JOB /B14805A / STOP  84124.1951 CPU    0MIN 00.98SEC SRB    0MIN 00.02SEC
+----------------------------------------------------------------------+
| PROCESSOR TIME ----------0.00036 CPU HOURS @ $1,260.00 ---------0.45 |
| PROCESSOR STORAGE -------0.13611 K-BYTE HOURS @  $0.50 ---------0.07 |
|                                  TOTAL PROCESSOR COST --------$0.52 |
|                                                                      |
| DISK EXCPS --------------------1 @  $0.36 PER 1000 -------------0.00 |
|             I/O COST (EXCLUDING PRINTER/READER/PUNCH) --------$0.00 |
|                                                                      |
| TOTAL COST (AFTER       $0.16   2ND SHIFT DISCOUNT) ----------$0.36 |
| AMOUNT OF FUNDS REMAINING ---------------------------------$106.44 |
| EXCLUDING CURRENT CHARGES FOR NON-COMPUTER SERVICES                  |
+----------------------------------------------------------------------+
*** NO ID ***          05/03/84 19:51      PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    *** NO ID ***
*** NO ID ***          05/03/84 19:51      PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    *** NO ID ***
*** NO ID ***          05/03/84 19:51      PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    *** NO ID ***
*** NO ID ***          05/03/84 19:51      PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    *** NO ID ***
*** NO ID ***          05/03/84 19:51      PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    *** NO ID ***
*** NO ID ***          05/03/84 19:51      PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    *** NO ID ***
*** NO ID ***          05/03/84 19:51      PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    *** NO ID ***
*** NO ID ***          05/03/84 19:51      PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    *** NO ID ***
*** NO ID ***          05/03/84 19:51      PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    *** NO ID ***
*** NO ID ***          05/03/84 19:51      PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    PL/C    *** NO ID ***

ERRORS/WARNINGS DETECTED DURING CODE GENERATION:

      WARNING: NO FILE SPECIFIED. SYSIN/SYSPRINT ASSUMED. (CGOC)
```



```
STRUCTURED TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM


EXAMPLE THREE
FROM  NEW TOPOLOGICAL FORMULA  SATYANARAYANA & PRABHAKAR
IEEE TRANS. ON RELIABILITY  PAGE 85  ARPA NETWORK
A COMPLEX SOURCE-TO-TERMINAL RELIABILITY NETWORK  CONTAINING
6 CYCLES.

OPTIONS USED
MATRIX LINKSET IMPORTANCE
```

```
STRUCTURED TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM

  NO. OF VERTICIES =   6  NO. OF EDGES =  12
START=         1          TERMINAL=         6;

INPUT DATA
   EDGE NO.  FROM      TO   RELIABILITY

      1        1        2    0.90           | 0 |  1 | 2 |  0 | 0 | 0 |
      2        1        3    0.90           | 0 |  0 | 9 |  4 | 8 | 0 |
      3        4        6    0.90           | 0 | 12 | 0 |  0 | 6 | 0 |
      4        2        4    0.90           | 0 |  0 | 0 |  0 | 7 | 3 |
      5        5        6    0.90           | 0 | 11 | 0 | 10 | 0 | 5 |
      6        3        5    0.90           | 0 |  0 | 0 |  0 | 0 | 0 |
      7        4        5    0.90
      8        2        5    0.90
      9        2        3    0.90
     10        5        4    0.90
     11        5        2    0.90
     12        3        2    0.90
```

```
STRUCTURED TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM


    1          2  5  6                              0.7290
   -1          2  3  5  6 10                       -0.5905
    1          2  3  6 10                           0.6561
   -1          1  2  5  6  8 12                     0.5314
   -1          2  5  6  8 12                       -0.5905
    1          2  5  8 12                           0.6561
   -1          1  2  5  8 12                       -0.5905
    1          1  5  8                              0.7290
   -1          1  2  5  6  8                       -0.5905
   -1          1  2  3  5  6  8 10 12              -0.4305
    1          2  3  5  6  8 10 12                  0.4783
```

```
STRUCTURED TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM

LINKSET
SIGN      SUBGRAPH                     RELIABILITY


    -1          2  3  6  8 10 12                  -0.5314
     1          2  3  8 10 12                      0.5905
    -1          2  3  5  8 10 12                  -0.5314
     1          1  2  3  6  8 10 12                0.4783
    -1          1  2  3  8 10 12                  -0.5314
     1          1  3  8 10                         0.6561
    -1          1  2  3  6  8 10                  -0.5314
     1          1  2  3  5  8 10 12                0.4783
    -1          1  3  5  8 10                     -0.5905
     1          1  2  3  5  6  8 10                0.4783
     1          1  2  3  4  5  6 10 11 12          0.3874
    -1          2  3  4  5  6 10 11 12            -0.4305
     1          2  3  4  6 10 11 12                0.4783
    -1          2  3  4  6 11 12                  -0.5314
     1          2  3  4 12                         0.6561
     1          2  3  4  6 11                      0.5905
    -1          2  3  4  6 10 12                  -0.5314
    -1          2  3  4  6 10 11                  -0.5314
     1          2  3  4  5  6 11 12                0.4783
    -1          2  3  4  5  6 12                  -0.5314
    -1          2  3  4  5  6 11                  -0.5314
     1          2  3  4  5  6 10 12                0.4783
```

```
STRUCTURED TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM

LINKSET
SIGN      SUBGRAPH                    RELIABILITY


     1            2  3  4  5  6 10 11                     0.4783
    -1            1  2  3  4  6 10 11 12                 -0.4305
     1            1  2  3  4  6 11 12                     0.4783
    -1            1  2  3  4 12                          -0.5905
     1            1  3  4                                 0.7290
    -1            1  2  3  4  6 11                       -0.5314
     1            1  2  3  4  6 10 12                     0.4783
    -1            1  2  3  4  6 10                       -0.5314
     1            1  2  3  4  6 10 11                     0.4783
    -1            1  2  3  4  5  6 11 12                 -0.4305
     1            1  2  3  4  5  6 12                     0.4783
    -1            1  2  3  4  5  6                       -0.5314
     1            1  2  3  4  5  6 11                     0.4783
    -1            1  2  3  4  5  6 10 12                 -0.4305
     1            1  2  3  4  5  6 10                     0.4783
    -1            1  2  3  4  5  6 10 11                 -0.4305
     1            1  2  3  4  5  6  8 10 12               0.3874
    -1            2  3  4  5  6  8 10 12                 -0.4305
     1            2  3  4  6  8 10 12                     0.4783
    -1            2  3  4  8 10 12                       -0.5314
     1            2  3  4  5  8 10 12                     0.4783
    -1            2  3  4  5  8 12                       -0.5314
```

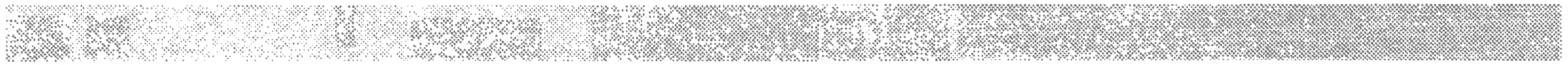
```
STRUCTURED TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM


LINKSET
SIGN       SUBGRAPH                    RELIABILITY



     1          2  3  4  5  6  8 12              0.4783
    -1          1  2  3  4  6  8 10 12          -0.4305
     1          1  2  3  4  8 10 12              0.4783
    -1          1  3  4  8 10                   -0.5905
     1          1  2  3  4  6  8 10              0.4783
    -1          1  2  3  4  5  8 10 12          -0.4305
     1          1  2  3  4  5  8 12              0.4783
    -1          1  3  4  5  8                   -0.5905
     1          1  3  4  5  8 10                 0.5314
    -1          1  2  3  4  5  6  8 12          -0.4305
     1          1  2  3  4  5  6  8              0.4783
    -1          1  2  3  4  5  6  8 10          -0.4305
     1          1  2  3  4  5  6  7  8 12        0.3874
    -1          2  3  4  5  6  7  8 12          -0.4305
     1          2  4  5  6  7  8 12              0.4783
    -1          2  4  5  7  8 12                -0.5314
     1          2  4  5  7 12                    0.5905
    -1          2  4  5  6  7 12                -0.5314
     1          2  3  4  5  7  8 12              0.4783
    -1          2  3  4  5  7 12                -0.5314
     1          2  3  4  5  6  7 12              0.4783
    -1          1  2  4  5  6  7  8 12          -0.4305
```

```
STRUCTURED TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM


LINKSET
SIGN      SUBGRAPH                   RELIABILITY



     1        1  2  4  5  7  8 12                  0.4783
    -1        1  2  4  5  7 12                    -0.5314
     1        1  4  5  7                           0.6561
    -1        1  4  5  7  8                       -0.5905
     1        1  2  4  5  6  7 12                  0.4783
    -1        1  2  4  5  6  7                    -0.5314
     1        1  2  4  5  6  7  8                  0.4783
    -1        1  2  3  4  5  7  8 12              -0.4305
     1        1  2  3  4  5  7 12                  0.4783
    -1        1  3  4  5  7                       -0.5905
     1        1  3  4  5  7  8                     0.5314
    -1        1  2  3  4  5  6  7 12              -0.4305
     1        1  2  3  4  5  6  7                  0.4783
    -1        1  2  3  4  5  6  7  8              -0.4305
     1        1  2  3  4  5  6  7  8  9            0.3874
    -1        1  3  4  5  6  7  8  9              -0.4305
     1        1  4  5  6  7  8  9                  0.4783
    -1        1  5  6  8  9                       -0.5905
     1        1  5  6  9                           0.6561
    -1        1  4  5  6  7  9                    -0.5314
     1        1  3  4  5  6  8  9                  0.4783
    -1        1  3  4  5  6  9                    -0.5314
```

```
STRUCTURED TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM

LINKSET
SIGN       SUBGRAPH                      RELIABILITY


   1          1  3  4  5  6  7  9             0.4783
  -1          1  2  4  5  6  7  8  9         -0.4305
   1          1  2  5  6  8  9                0.5314
  -1          1  2  5  6  9                  -0.5905
   1          1  2  4  5  6  7  9             0.4783
  -1          1  2  3  4  5  6  8  9         -0.4305
   1          1  2  3  4  5  6  9             0.4783
  -1          1  2  3  4  5  6  7  9         -0.4305
   1          1  2  3  4  5  6  8  9 10       0.3874
  -1          1  3  4  5  6  8  9 10         -0.4305
   1          1  3  5  6  8  9 10             0.4783
  -1          1  3  6  8  9 10               -0.5314
   1          1  3  6  9 10                   0.5905
  -1          1  3  5  6  9 10               -0.5314
   1          1  3  4  6  8  9 10             0.4783
  -1          1  3  4  6  9 10               -0.5314
   1          1  3  4  5  6  9 10             0.4783
  -1          1  2  3  5  6  8  9 10         -0.4305
   1          1  2  3  6  8  9 10             0.4783
  -1          1  2  3  6  9 10               -0.5314
   1          1  2  3  5  6  9 10             0.4783
  -1          1  2  3  4  6  8  9 10         -0.4305
```

```
STRUCTURED TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM

LINKSET
SIGN      SUBGRAPH                     RELIABILITY


    1           1  2  3  4  6  9 10                    0.4783
   -1           1  2  3  4  5  6  9 10                -0.4305
                                                      ________
GRAPH RELIABILITY                                     0.977184
```

```
STRUCTURED TOPOLOGICAL RELIABILITY ANALYSIS PROGRAM


EDGE IMPORTANCE

IMPORTANCE( 1) =  0.107580
IMPORTANCE( 2) =  0.098678
IMPORTANCE( 3) =  0.098678
IMPORTANCE( 4) =  0.019217
IMPORTANCE( 5) =  0.107580
IMPORTANCE( 6) =  0.019217
IMPORTANCE( 7) =  0.000860
IMPORTANCE( 8) =  0.011278
IMPORTANCE( 9) =  0.000860
IMPORTANCE(10) =  0.008872
IMPORTANCE(11) =  0.000066
IMPORTANCE(12) =  0.008872
```



```
IN STMT  525  PROGRAM RETURNS FROM MAIN PROCEDURE.
```

```
1/**************************************************************/   00000100   PL/C-R7   POST-MORTEM DUMP   PAGE    1

IN STMT  525  SCALARS AND BLOCK-TRACE:

***** MAIN PROCEDURE MAIN

H= 5.00000E+00          VERTX= 0.00000E+00        REL= ?.?????E+??        OPTIONS='IMPORTANCE '    COMMENTS='
                                   '              T_FLAG='0'B             I_FLAG='1'B              L_FLAG='1'B
M_FLAG='1'B             D_FLAG='1'B               C_FLAG='0'B             LINE=       15           M=       12
N=       6

NON-0 PROCEDURE EXECUTION COUNTS:
NAME         STMT COUNT  NAME         STMT COUNT  NAME         STMT COUNT  NAME         STMT COUNT  NAME         STMT COUNT
LINKSET      0509 00006  RULE_FOUR    0248 00123  RELIABILITY 0388 00123  RESTORE      0354 00281  RULE_THREE   0216 00009
P_GRAPH      0368 00210  RULE_TWO     0185 00015  REMOVE       0292 00287  SEARCH       0144 00249  RULE_ONE     0132 00029
DATA_IN      0429 00001  HEADER       0518 00010  MAIN         0001 00001


      COMPILATION STATISTICS  (0525 STATEMENTS)  |                     EXECUTION STATISTICS
SECONDS  ERRORS  WARNINGS  PAGES  LINES  CARDS  INCL'S | SECONDS  ERRORS  WARNINGS  PAGES  LINES  CARDS  INCL'S  AUX I/O
    .30      0         1      0      4    609      0 |     .63       0         0     12    427      0       0       19
-----------------------------------------------------+-----------------------------------------------------------------
BYTES    SYMBOL TABLE    INTERMEDIATE CODE    OBJECT CODE |    STATIC CORE    AUTOMATIC CORE     DYNAMIC CORE   TOTAL STORAGE
USED     12204(  12K)      13758(  14K)      30118(  30K) |      340(   1K)     13306(  13K)     15156(  15K)    70070(  69K)
UNUSED    4596( 102K)     158313( 154K)     247982( 242K) |   219214( 214K)    219214( 214K)   158313( 154K)   219214( 214K)

AUXILIARY I/O:

 INPUT:     19 RECORDS FROM INPUT

THIS PROGRAM MAY BE RERUN WITHOUT CHANGE IN A REGION 214K BYTES SMALLER USING TABLESIZE=  3051
```