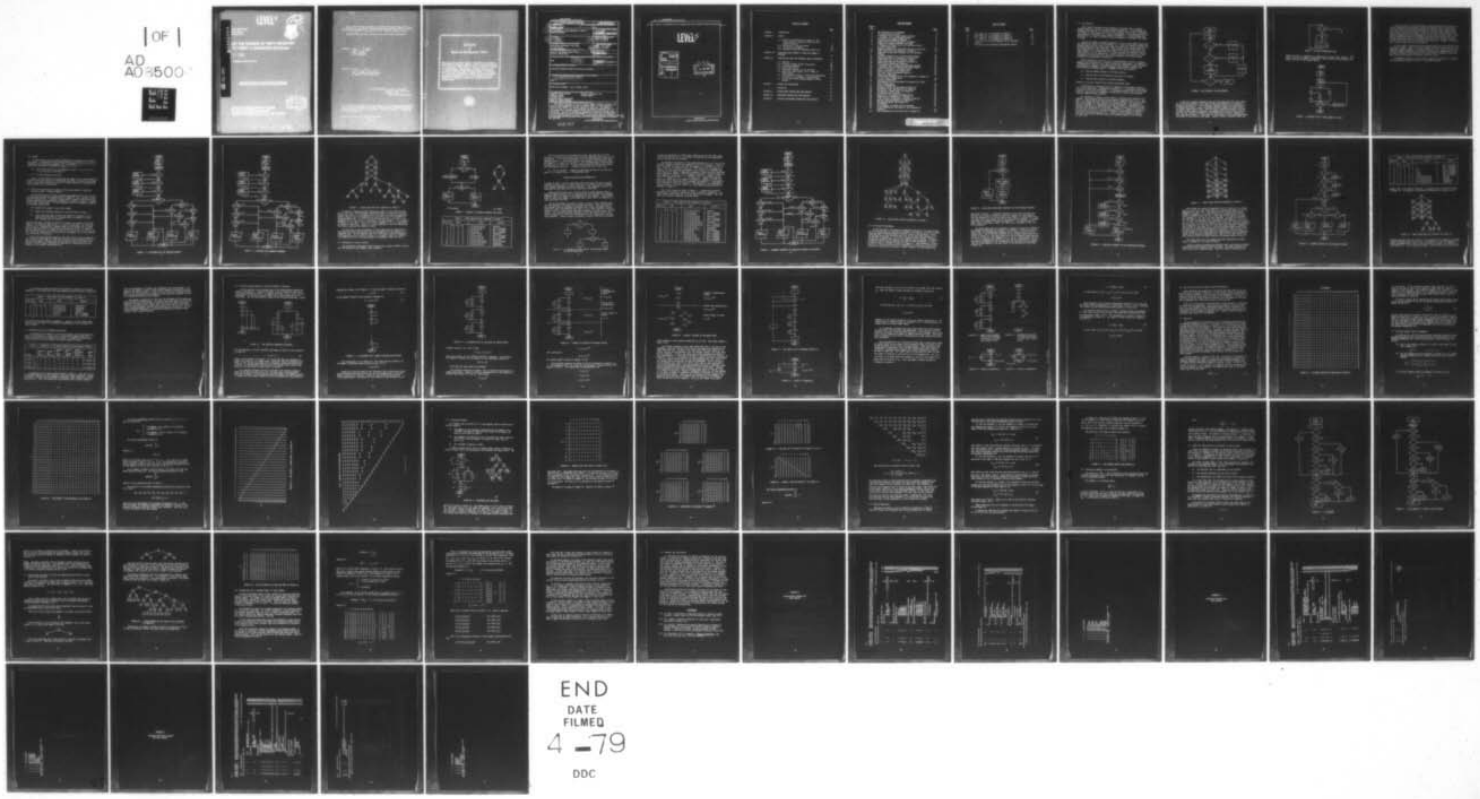
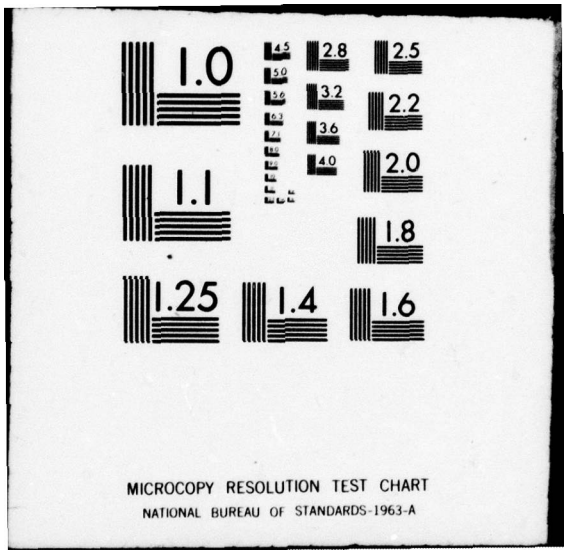
AD-A065 004 POLYTECHNIC INST OF NEW YORK BROOKLYN DEPT OF ELECTR--ETC F/G 9/2
ON THE NUMBER OF TESTS NECESSARY TO VERIFY A COMPUTER PROGRAM.(U)
NOV 78 G S POPKIN, M L SHOOMAN F30602-78-C-0057
UNCLASSIFIED POLY-EE78-047 RADC-TR-78-229 NL

| OF |
AD
A065004



END
DATE
FILMED
4 -79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

LEVEL II





RADC-TR-78-229
Interim Report
November 1978

# ON THE NUMBER OF TESTS NECESSARY TO VERIFY A COMPUTER PROGRAM

G. S. Popkin
M. L. Shooman

Polytechnic Institute of NY





[redacted]

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-78-229 has been reviewed and is approved for publication.

APPROVED: Alan N Sukert

ALAN N. SUKERT
Project Engineer

APPROVED: Alan R Barnum

ALAN R. BARNUM, Assistant Chief
Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIS) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

# MISSION
# of
# Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications ($C^3$) activities, and in the $C^3$ areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

## REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

| | |
|---|---|
| 1. REPORT NUMBER: RADC-TR-78-229 | 2. GOVT ACCESSION NO. |
| 3. RECIPIENT'S CATALOG NUMBER | |
| 4. TITLE (and Subtitle): ON THE NUMBER OF TESTS NECESSARY TO VERIFY A COMPUTER PROGRAM | 5. TYPE OF REPORT & PERIOD COVERED: Interim Report, Jan [redacted] - August 1978. |
| | 6. PERFORMING ORG. REPORT NUMBER: Poly EE 78-047 |
| 7. AUTHOR(s): G. S. Popkin, M. L. Shooman | 8. CONTRACT OR GRANT NUMBER(s): F30602-78-C-0057 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS: Polytechnic Institute of New York, 333 Jay Street, Brooklyn, NY 11201 Dept. of Electrical Engineering and Electrophysics | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F, 23040401, J4 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIS), Griffiss AFB NY 13441 | 12. REPORT DATE: November 1978 |
| | 13. NUMBER OF PAGES: 67 |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same (12) 72 Pgs | 15. SECURITY CLASS. (of this report): UNCLASSIFIED |
| | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

(14) POLY-EE78-047

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Alan N. Sukert (ISIS)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Program Testing
Flowchart Graphs
Adjacency Matrix
Zero-One Linear Programming
Level One Program Testing
Maximum Incomparable Sets
Program Segment

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report discusses the use of flowchart graphs, adjacency matrices, and zero-one linear programming to find the minimum number of tests necessary to execute every segment of a computer program at least once. The methods of Lipow are used as the basis for determining the maximum incomparable set, i.e., the largest set of program segments through which one and only one test case should pass. The size of the maximum incomparable set gives the minimum number of tests necessary to execute each segment at least once, while the elements of this set give the paths of each test. The report develops methods for finding

next page

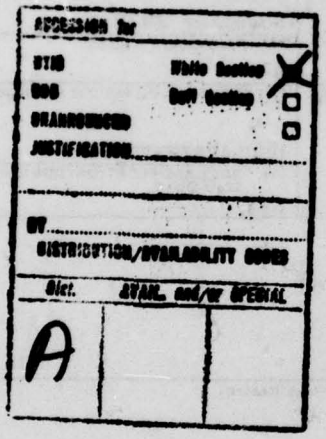
DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

cont. the maximum incomparable set for loopless and some elementary looping flowcharts.

LEVEL II



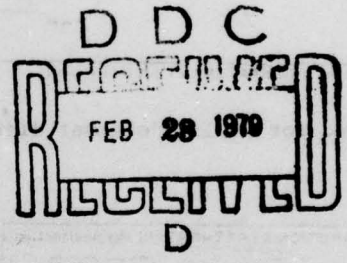



TABLE OF CONTENTS

## LIST OF FIGURES





## LIST OF TABLES

## 1.0 Introduction

This report discusses various aspects of verifying that a computer program correctly carries out some specified functions. If the program was designed with the aid of a flowchart, the flowchart can be used to determine the number of tests necessary to verify the program. If the program was prepared without a flowchart, then either a flowchart or a directed graph (as described in 2.0) must be prepared from the program, to determine the number of tests.

One way to verify a computer program is to run it with sample input data and examine the results for correctness, i.e., test for agreement between the program output and the results that would be produced by correct execution of the specification on the same data. This requires prior determination via hand computation or other independent means of the correct outputs for the sample inputs. It is useful for us to first classify the different types of tests which we will be discussing. A level zero test is defined as a test consisting of one or more test cases which together cause every program statement to be executed at least once.

Before defining a level one test, it is necessary to define a program segment, which we shall do in terms of flowchart terminology. In general our flowcharts will consist of one start and one stop terminal (ovals), processing elements (rectangles or parallelograms), and decision elements also called deciders (diamond shape). A segment is any flow sequence:

a) from the START terminal to the first decider,

b) from the exit of one decider to the entry of another,

c) from the last decider to the STOP terminal,

whether or not any of these flows contain processing steps.

A subroutine is considered part of a segment if the segment contains a call to the subroutine. Segments may overlap, i.e., a processing step may be contained in more than one segment. If a program contains more than one STOP terminal, then a group of segments is formed by the flow to each terminal from the respective last decider.

The flowchart of Figure 1 can be used to illustrate the definition of a segment. The flowchart contains seven segments, numbered from 0 to 6. Segment 1 is clearly the flow from the exit of one decider to the entry of another, even though the flow contains no processing steps. The subroutine called in segment 3 is completely contained in segment 3, regardless of any deciders that appear in the subroutine. Segment 2 consists of all the flow from the exit of decider A to the entry of decider C. Segment 4 consists of all the flow from B to C, and part of segment 4 overlaps part of segment 2 (they share the processing step 3). A level one test consists of one or more test cases which together cause every program segment to be executed at least once.

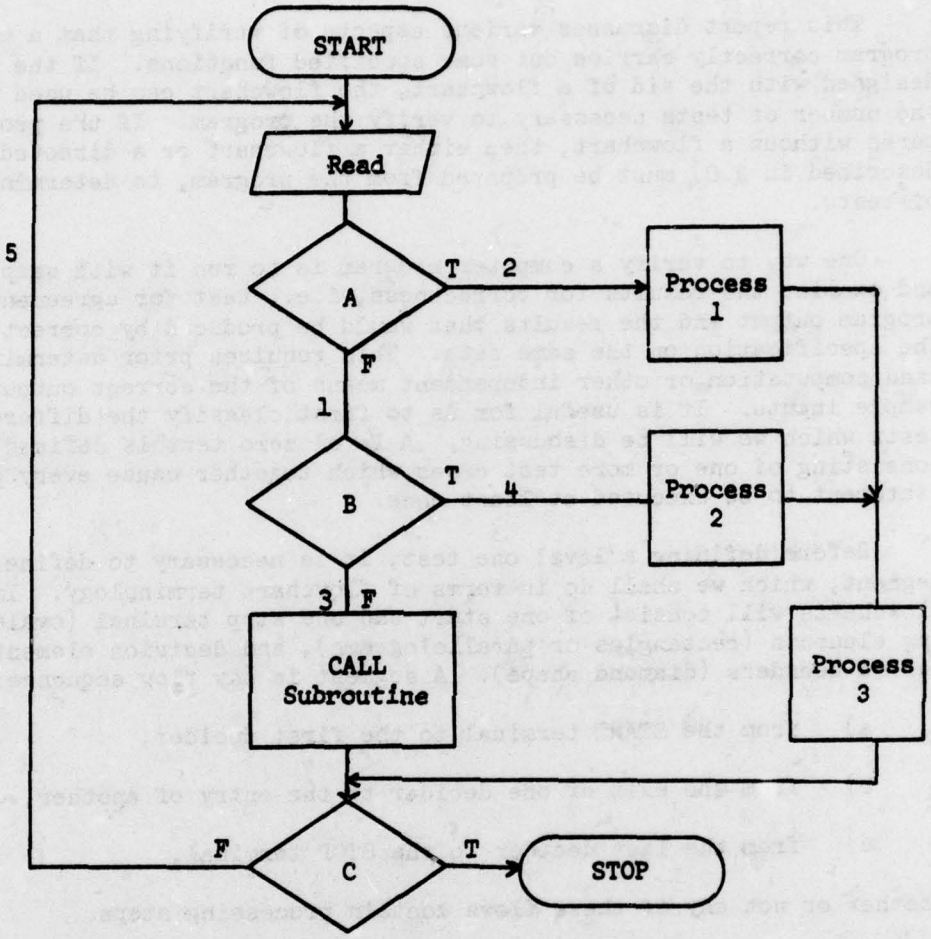



FIGURE 1 THE FLOWCHART FOR THE EXAMPLE.

A path in a computer program is any way of getting from the start terminal to the stop terminal, or to each of the stop terminals, if the program contains more than one. If the flowchart contains loops, each different way of getting from start to stop, including different numbers of times around the loops, constitutes a path. This may lead to situations where the number of paths is very large or difficult to determine from the flowchart, such as when the loop control is defined in some complex way or when it depends on input data as in Figure 2. Even when the loop control is more straightforward, the number of paths can be very large. Figure 3 shows a flowchart where the

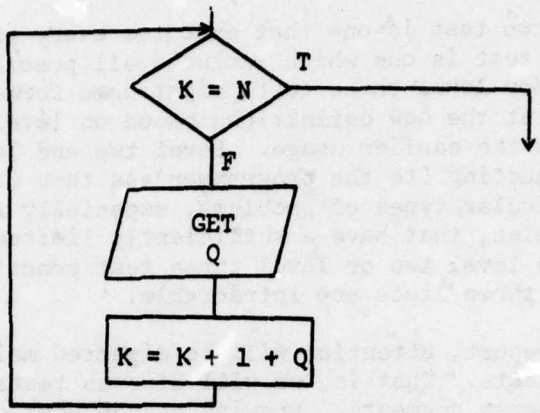



FIGURE 2 AN INPUT-CONTROLLED LOOP.

number of paths, p, depends on a single value of input data, where $p = 3^{N+1}$. Since N can be the largest integer that can be stored in the computer, the number of paths is very large indeed.

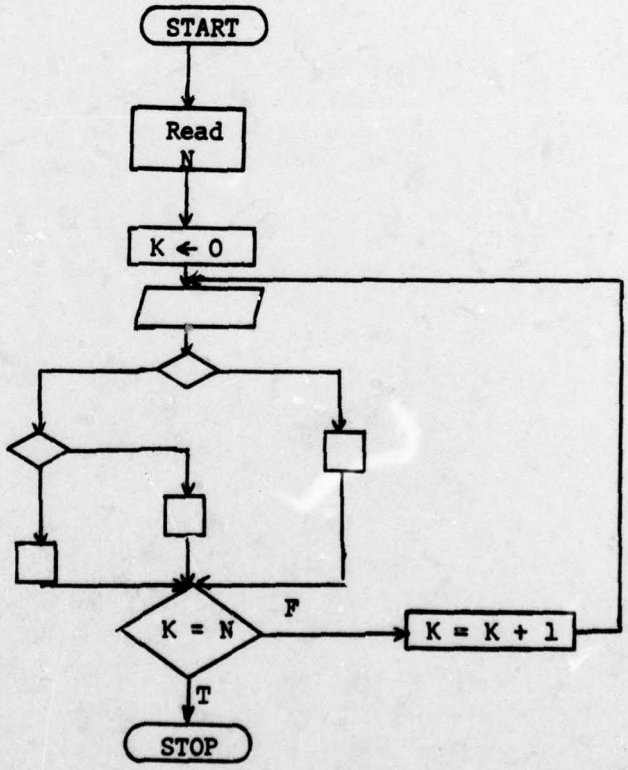



FIGURE 3 A FLOWCHART WITH A LARGE NUMBER OF PATHS.

A level two test is one that executes every program path at least once. A level three test is one which includes all possible sets of input data. Level two and/or level three tests might have formerly been called "exhaustive tests," but the new definitions based on levels are more precise and should replace the earlier usage. Level two and level three tests are not always as exhausting (to the programmer) as they might at first seem. There are some particular types of problems, especially those dealing only with integer variables, that have a sufficiently limited set of input possibilities to make a level two or level three test practical. However, in general level two and three lists are intractable.

In this report, attention will be directed mainly to level one and related types of tests. That is, we will discuss tests which are oriented to the testing of program segments. Drawing on the work of M. Lipow [1], it will be shown how graphs, matrices, and zero-one integer linear programming [4] can be used to determine the number of test cases needed to perform a level one test, and the data needed to comprise those test cases.

In addition, Section 3 of this report contains a discussion of methods of estimating the number of paths in a loopless flowchart.

## 2.0 Graphs

In this section we will examine techniques for computing the number of test cases. As examples we will consider the testing of one unstructured program and two structured programs. For the purpose of this report, it is sufficient to define a structured program as one in which:

a) each major function of the program is coded in its own "module," (i.e., function or subroutine),

b) no GOTO statements are used.

First, a formal method of determining the number of test cases needed to perform a level one test will be described; secondly, the method will be applied to the three programs in question; finally, experience in testing the three programs will be discussed.

### 2.1 Method of Determining the Number of Test Cases Needed to Perform a Level One Test -- (Lipow's Method).

For the description of Lipow's method using graphs [1], we will use the flowchart given by him and repeated here in Figure 4. The flowchart solves the following problem: "Determine whether three integers representing three lengths constitute an equilateral, isosceles, or scalene triangle, or cannot be the sides of any angle." In this flowchart we will be seeking to execute each segment exactly once.

To determine the number of test cases needed:

a) number the flowchart segments, as has been done in Figure 5,

b) draw a directed graph in which the nodes of the graph are the segments and the arcs of the graph show the sequence in which the segments may be executed.

This has been done in Figure 6, where it is shown, for example, that after segment 6 is executed, then either 7 or 8 will be executed. Now, find the maximum set of "incomparable" elements; that is, the largest set of nodes none of which precedes or follows any other node in the set. In Figure 6, the set consists of the 11 nodes 24, 23, 17, 11, 16, 15, 22, 21, 26, 25, and 19. These all happen to be at the bottom of the graph, but we will see that this is merely a coincidence.

After the graph has been analyzed, the size of the largest incomparable set is equal to the number of test cases required, and the elements of the set are the segments through which one and only one test case should pass. Section 4 of this report describes an algorithm for finding the size and members of the maximum incomparable set.

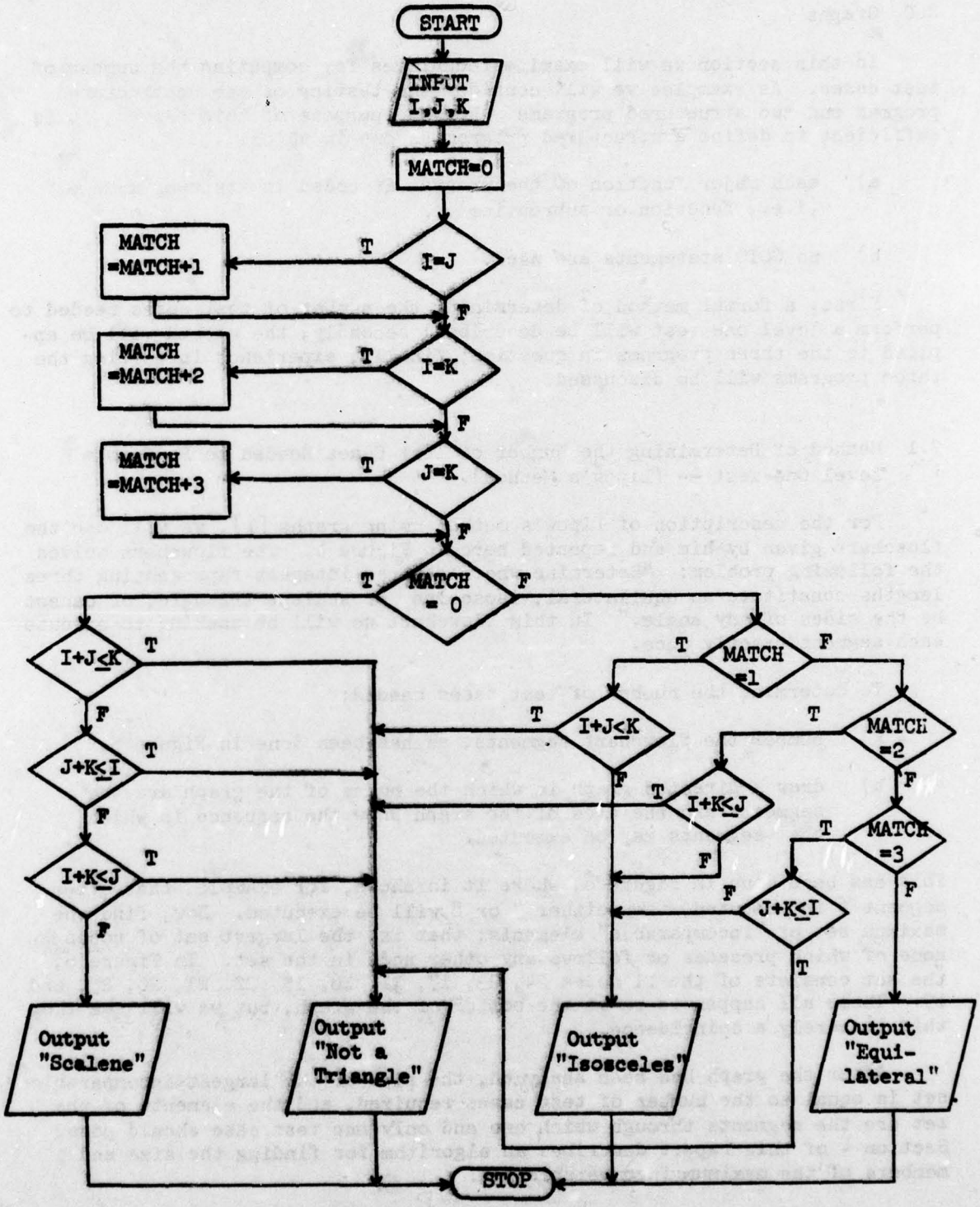



FIGURE 4 -- A FLOWCHART FOR THE TRIANGLE PROBLEM

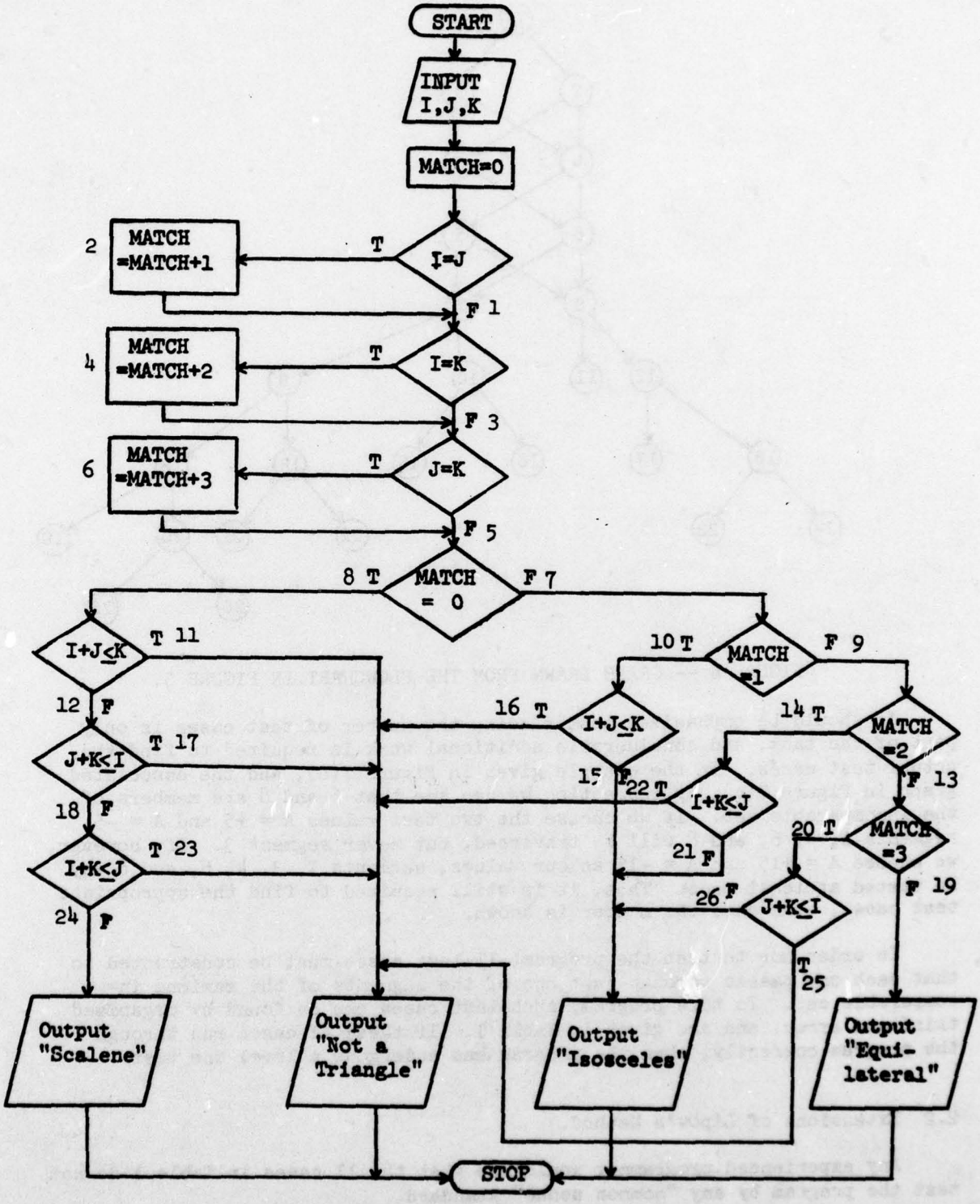



FIGURE 5 -- FLOWCHART WITH SEGMENTS NUMBERED.

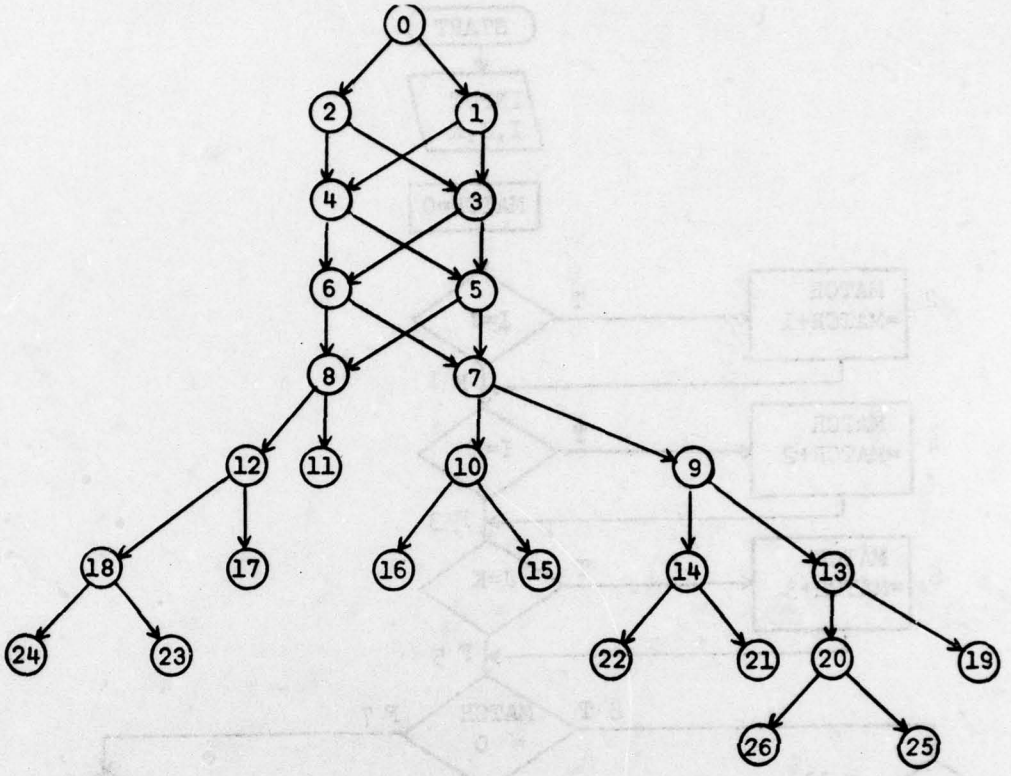


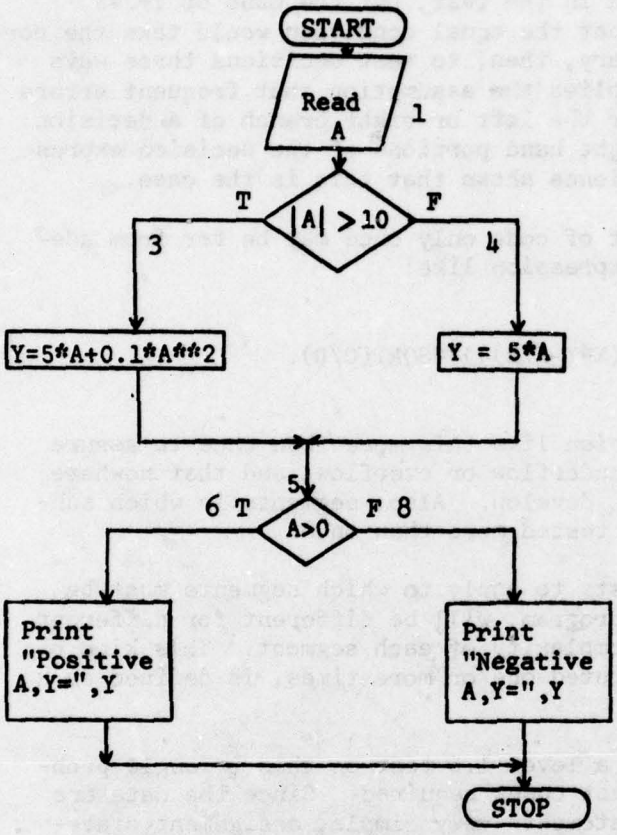

FIGURE 6 -- GRAPH DRAWN FROM THE FLOWCHART IN FIGURE 5.

It should be emphasized that finding the number of test cases is only part of the task, and considerable additional work is required to find the actual test cases. In the example given in Figure 7(a), and the associated graph in Figure 7(b), by inspection we can see that 6 and 8 are members of the incomparable set. If we choose the two test values A = +5 and A = -5, segments 1, 4, 6, and 8 will be traversed, but never segment 3. If, however, we choose A = +15 and A = -15 as our values, segments 1, 3, 4, 6, and 8 may be tested at least once. Thus, it is still required to find the appropriate test cases, even once the member is known.

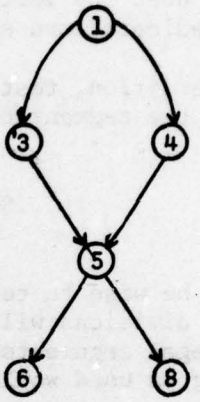
In order now to test the program, 11 test cases must be constructed so that each one passes through just one of the segments of the maximum incomparable set. In this program, such test cases can be found by organized trial and error, and are given in Table 1. If these 11 cases run through the program correctly, then the program has undergone a level one test.

## 2.2 Extensions of Lipow's Method.

Any experienced programmer would see that the 11 cases in Table 1 do not test the program by any "common sense" standard.




(a) Flowchart




(b) Graph

FIGURE 7 -- EXAMPLE OF PROGRAM FLOWCHART AND GRAPH.

TABLE 1 -- TEST CASES FOR THE FLOWCHART IN FIGURE 5.

| Test No. | Data | Path | Results |
|---|---|---|---|
| 1 | I=4 J=3 K=2 | 1-3-5-8-12-18-24 | Scalene |
| 2 | 2 4 1 | 1-3-5-8-12-18-23 | Not a triangle |
| 3 | 4 1 2 | 1-3-5-8-12-17 | Not a triangle |
| 4 | 1 2 4 | 1-3-5-8-11 | Not a triangle |
| 5 | 2 2 3 | 2-3-5-7-10-16 | Isosceles |
| 6 | 2 2 5 | 2-3-5-7-10-15 | Not a triangle |
| 7 | 2 5 2 | 1-4-5-7-9-14-22 | Not a triangle |
| 8 | 2 3 2 | 1-4-5-7-9-14-21 | Isosceles |
| 9 | 5 2 2 | 1-3-6-7-9-13-20-26 | Not a triangle |
| 10 | 3 2 2 | 1-3-6-7-9-13-20-25 | Isosceles |
| 11 | 1 1 1 | 2-4-6-7-9-13-19 | Equilateral |

The case of I+J<K was included in the test, but the case of I+J=K was not, so we have no assurance that the equal condition would take the correct branch. It is clearly necessary, then, to test decisions three ways whenever possible. This really implies the assumption that frequent errors are committed in specifying whether the left or right branch of a decision is taken when the left hand and right hand portions of the decision expression (predicate) are equal. Experience shows that this is the case.

In addition, testing a segment of code only once may be far from adequate if the segment contains an expression like:

```
SQRT(SIN(ALOG(X*Y-A/B)))**SQRT(C/D).
```

It might be wise to test an expression like this more than once to assure that the divisions will not cause underflow or overflow, and that nowhere will illegal arguments of functions develop. Also, segments in which subscripting is used would have to be tested more than once.

These decisions about what tests to apply to which segments must be made by someone familiar with the program, will be different for different programs, and will depend on the complexity of each segment. This kind of test, in which each segment is executed one or more times, is defined as a level one-point-five test.

Let us now attempt to perform a level 1.5 test on this triangle problem, and determine the number of test cases required. Since the data are all integers and the assignment statements very simple, assignment statements will need to be executed only once, as in a level one test. But all decisions must be tested three ways where possible, for high, equal, and low values. For certain decisions, however, we will see that no set of test data can exercise all three outcomes because not all combinations are logically possible. Figure 8 shows a flowchart in which it may not be possible

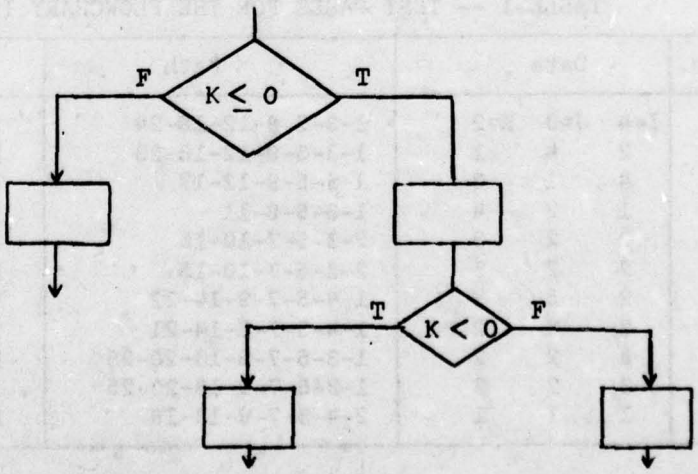



FIGURE 8 -- A FLOWCHART IN WHICH ONE OF THE DECISIONS CANNOT BE TESTED THREE WAYS.

to test the decision K < 0 three ways, since no set of test data could reach that decision with K > 0 unless K is modified by the unspecified process between the two deciders.

To determine the number of test cases required for a 1.5 test on the triangle problem, renumber the segments, this time assigning to each segment one number for each time that the segment is to be executed in the test. That is, to branches which are to be taken twice we assign two numbers; if there had been any segments that needed to be executed more than once due to complex assignment statements, these too would have been given additional numbers. The renumbered flowchart is shown in Figure 9, where segment 1 stands for I < J, segment 2 for I > J, segment 12 for I + J < K, segment 13 for I + J = K, and so on. Segment 22 stands for MATCH > 1; there can be no segment for MATCH < 1, since no set of test data could reach that point in the program with MATCH < 1. The graph of this flowchart is shown in Figure 10. The maximum set of incomparable elements numbers is 17,and consists of those segments appearing with an asterisk in Figure 10.

The 17 test cases are shown in Table 2. A conventional program realizing this flowchart, written in PL/C, is given in Appendix A. The output shown results from the 17 cases that test the program. The test cases were found by organized trial and error.

TABLE 2 -- TEST CASES FOR THE FLOWCHART IN FIGURE 9.

| Test No. | Data | | | Path | Results |
|---|---|---|---|---|---|
| 1 | I=4 | J=3 | K=2 | 3-6-9-11-14-15-18 | Scalene |
| 2 | 1 | 1 | 3 | 1-5-8-10-21-23 | Not a triangle |
| 3 | 1 | 1 | 2 | 1-5-8-10-21-24 | Not a triangle |
| 4 | 2 | 3 | 2 | 2-4-9-10-22-26-28 | Isosceles |
| 5 | 2 | 5 | 2 | 2-4-9-10-22-26-29 | Not a triangle |
| 6 | 2 | 4 | 2 | 2-4-9-10-22-26-30 | Not a triangle |
| 7 | 1 | 2 | 2 | 2-5-7-10-22-27-31-35 | Isosceles |
| 8 | 5 | 2 | 2 | 3-6-7-10-22-27-31-33 | Not a triangle |
| 9 | 4 | 4 | 4 | 3-6-7-10-22-27-31-34 | Not a triangle |
| 10 | 3 | 3 | 2 | 1-6-9-10-21-25 | Isosceles |
| 11 | 1 | 1 | 1 | 1-4-7-10-22-27-32 | Equilateral |
| 12 | 0 | 1 | 2 | 2-5-8-11-12 | Not a triangle |
| 13 | 1 | 2 | 3 | 2-5-8-11-13 | Not a triangle |
| 14 | 2 | 0 | 1 | 3-6-8-11-14-16 | Not a triangle |
| 15 | 3 | 1 | 2 | 3-6-8-11-14-17 | Not a triangle |
| 16 | 1 | 2 | 0 | 2-6-9-11-14-15-19 | Not a triangle |
| 17 | 2 | 3 | 1 | 2-6-9-11-14-15-20 | Not a triangle |

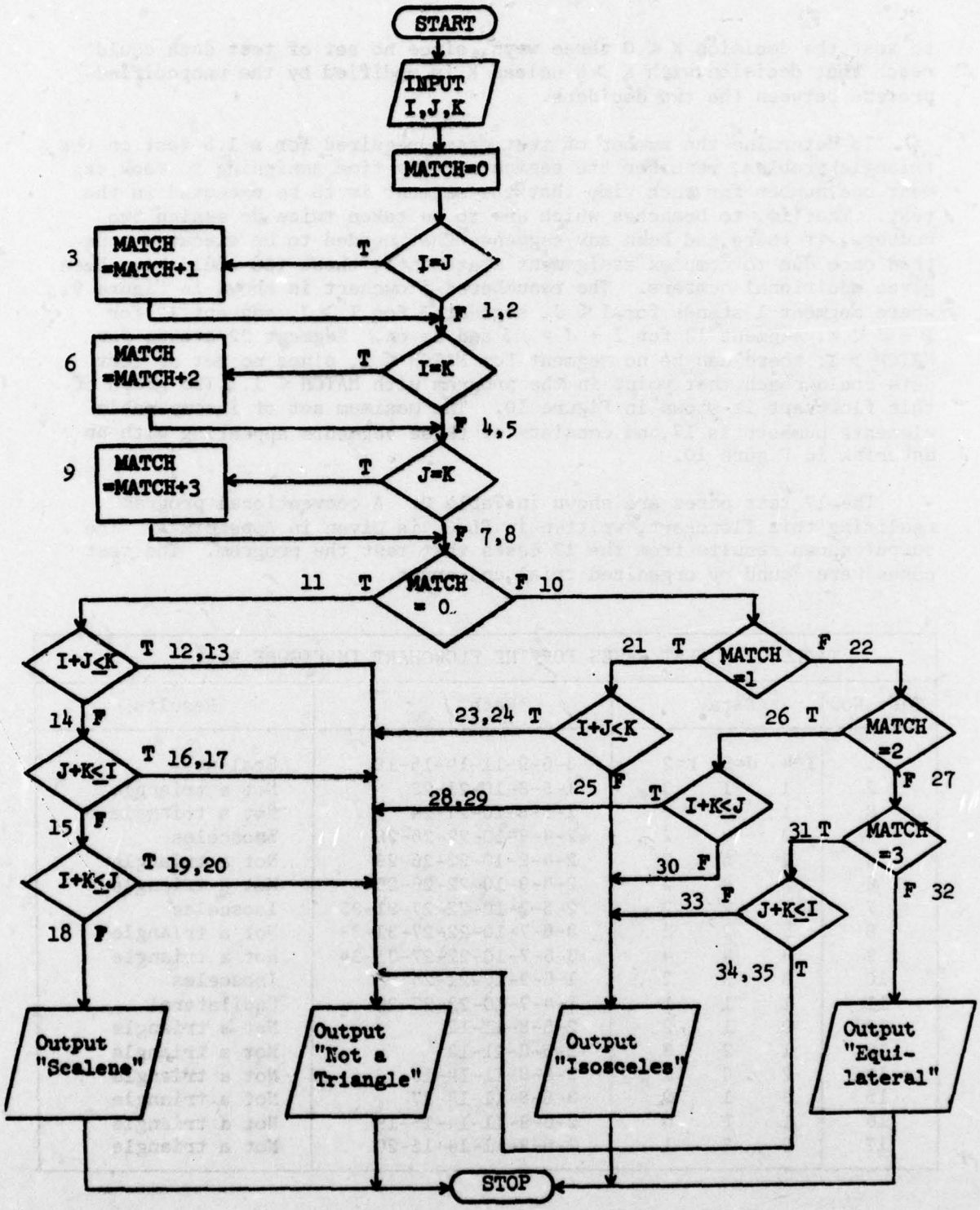



FIGURE 9 -- SEGMENTS NUMBERED FOR THREE-WAY TESTING OF DECISIONS.

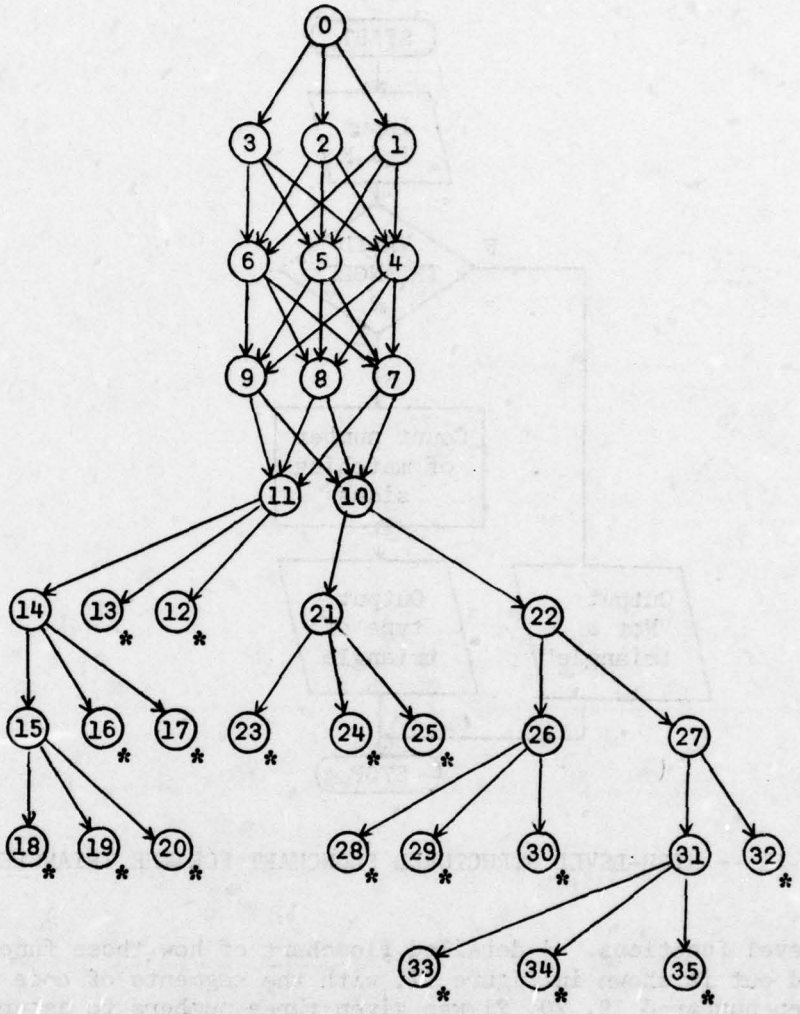



FIGURE 10 -- GRAPH DRAWN FROM THE FLOWCHART IN FIGURE 9.

## 2.3 Structured Programming

We may now take a "structured programming" approach to the same problem, by first noting that the flowchart of Figure 4 has frequent crossing of flowlines and much branching between different areas of the flowchart. These characteristics of the flowchart give rise to the many GOTO statements in the program of Appendix A. We will now attempt a new flowchart, trying to isolate the different functions of the program from one another, in order to eventually produce easily readable code free of GOTO statements. A first attempt at such a flowchart is shown in Figure 11. This flowchart is drawn as if for a virtual machine having a capability of executing a decision on whether the input data constitute a "valid triangle," and performing other

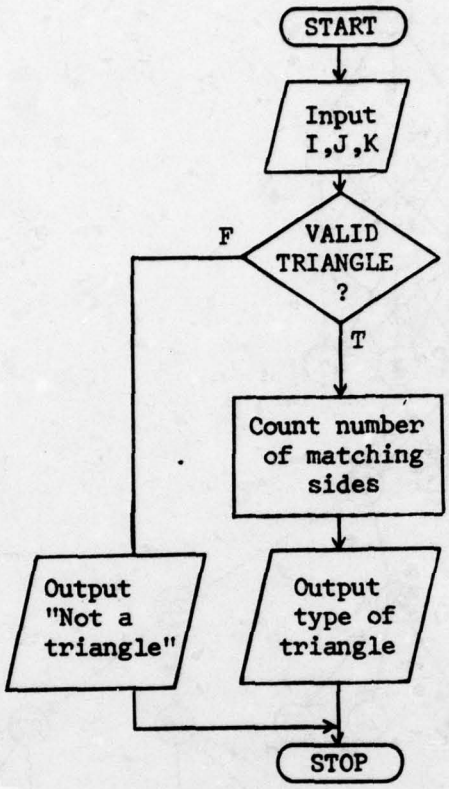



FIGURE 11 -- HIGH-LEVEL STRUCTURED FLOWCHART FOR THE TRIANGLE PROBLEM.

high-level functions. A detailed flowchart of how those functions are carried out is shown in Figure 12, with the segments of code numbered. The step numbered 19, 20, 21 was given three numbers to assure that three test cases will pass through it, one for each type of triangle. A program realizing this flowchart is given in Appendix B. The program uses IF... THEN...ELSE, a function, a CALL, a DO loop, and a DO group to avoid all GOTO statements.

The graph drawn from the flowchart in Figure 12 is given in Figure 13. The maximum incomparable set consists of 9 elements. There are four different sets of 9 incomparable elements, and we can use any of the four equally well. One of the sets consists of the elements marked with an asterisk in Figure 13. The other three sets can be formed by replacing segments 10, 11, and 12, with segments 13, 14, and 15, respectively; with 16, 17, and 18, respectively; and with 19, 20, and 21, respectively. In all four cases the members of the incomparable set are pair-wise unreachable from all other members of the set, which is the definition of an incomparable set.

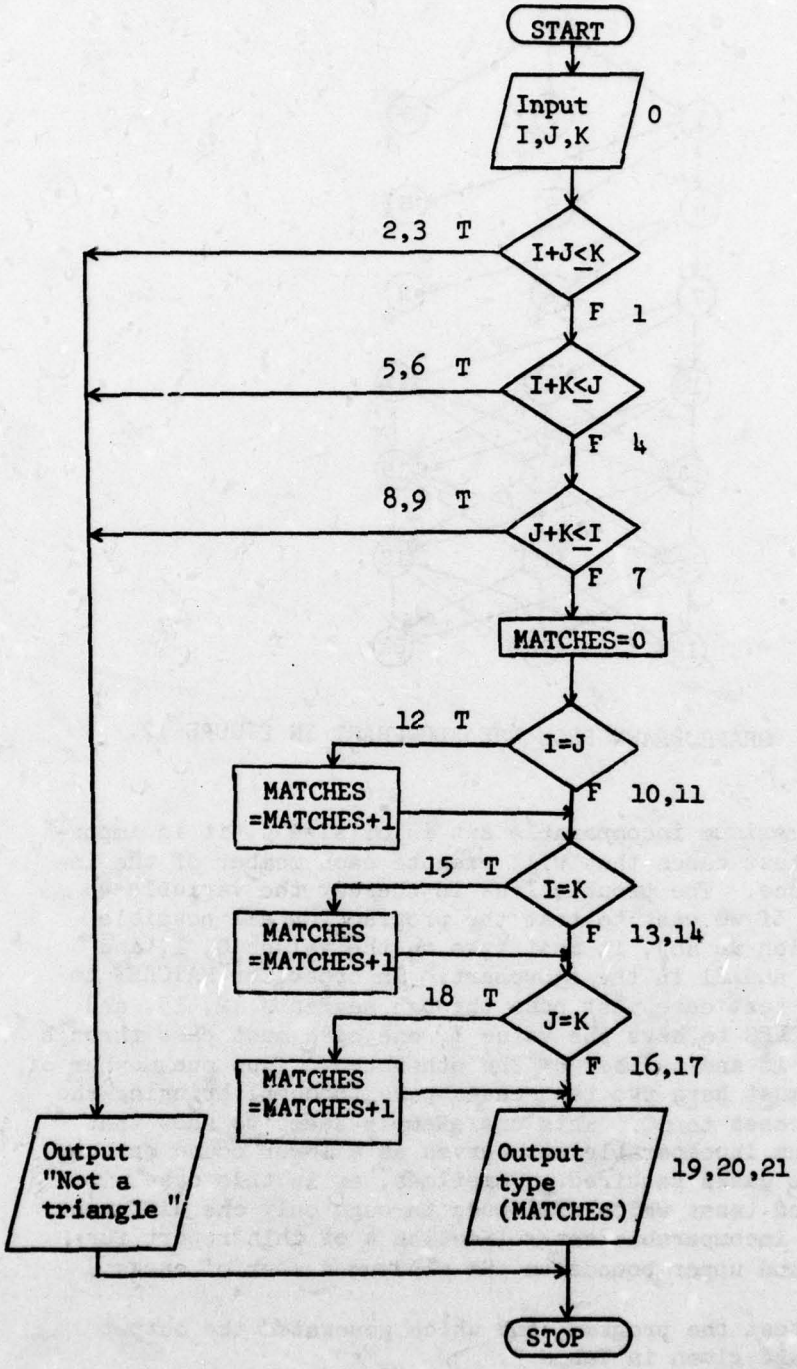



FIGURE 12 -- DETAILED FLOWCHART FOR THE STRUCTURED PROGRAM.

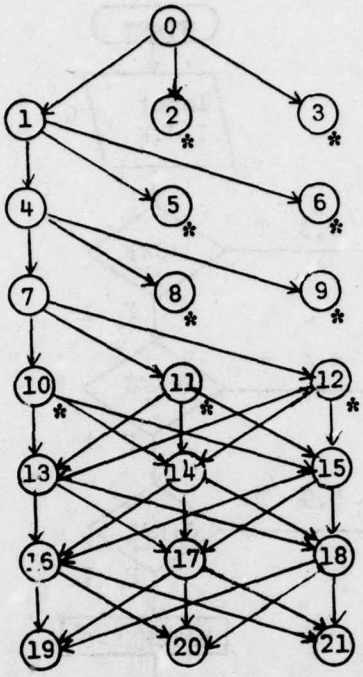



FIGURE 13 -- GRAPH DRAWN FROM THE FLOWCHART IN FIGURE 12.

Even though the maximum incomparable set is of size 9, it is impossible to find only 9 test cases that will execute each member of the incomparable set only once. The trouble lies in the way the variable MATCHES is computed. If we want to test the program for all possible values of MATCHES (which we do), it must take on the values 0, 1, and 3 at segments 19, 20, and 21 in the flowchart. In order for MATCHES to have the value 3, one test case must pass through segments 12, 15, and 18. In order for MATCHES to have the value 1, one case must pass through any one of 12, 15, or 18 and neither of the other two. Thus one member of the incomparable set must have two test cases pass through, bringing the total number of test cases to 10. This one example seems to show that the size of the maximum incomparable set serves as a lower bound on the minimum number of test cases required. Sometimes, as in this case, it is not possible to find tests which will pass through only one different member of the maximum incomparable set. (Section 4 of this report further discusses lower and upper bounds on the minimum number of cases.)

Ten cases which test the program, and which generated the output shown in Appendix B, are given in Table 3.

Figure 14 shows an essentially different way of solving the triangle problem, one which requires fewer test cases. The input variables are first sorted into ascending order. Then the required comparisons are

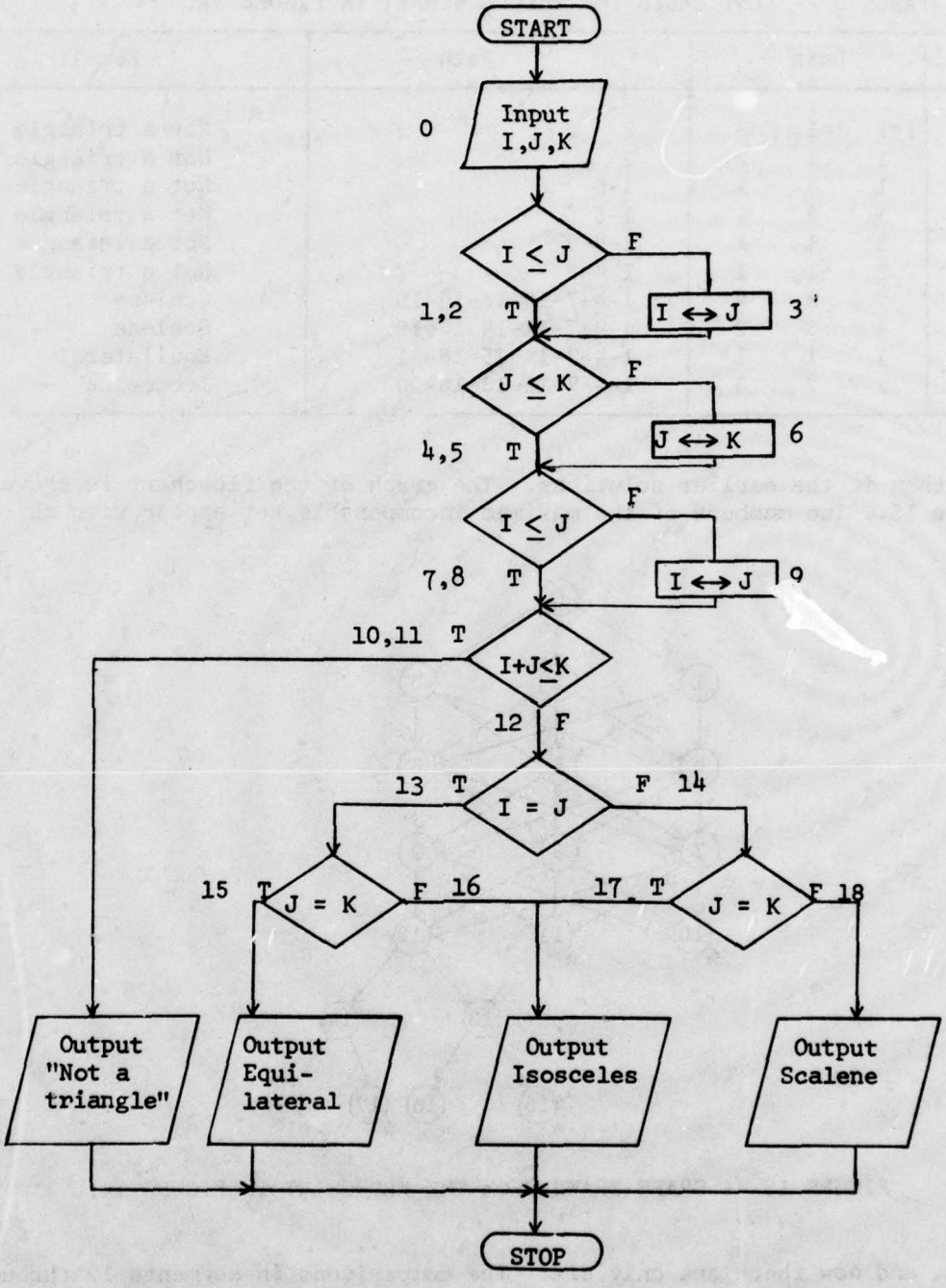



FIGURE 14 -- ANOTHER SOLUTION TO THE TRIANGLE PROBLEM.

| TABLE 3 -- TEST CASES FOR THE FLOWCHART IN FIGURE 12. | | | | | |
|---|---|---|---|---|---|
| Test No. | Data | | | Path | Results |
| 1 | I=1 | J=1 | K=3 | 3 | Not a triangle |
| 2 | 1 | 1 | 2 | 2 | Not a triangle |
| 3 | 1 | 3 | 1 | 1-6 | Not a triangle |
| 4 | 1 | 2 | 1 | 1-5 | Not a triangle |
| 5 | 3 | 1 | 1 | 1-4-9 | Not a triangle |
| 6 | 2 | 1 | 1 | 1-4-8 | Not a triangle |
| 7 | 2 | 3 | 4 | 1-4-7-10-13-16-19 | Scalene |
| 8 | 4 | 3 | 2 | 1-4-7-11-14-17-19 | Scalene |
| 9 | 1 | 1 | 1 | 1-4-7-12-15-18-21 | Equilateral |
| 10 | 2 | 2 | 3 | 1-4-7-12-13-16-20 | Isosceles |

simpler than in the earlier solutions. The graph of the flowchart is shown in Figure 15. The members of the maximum incomparable set appear with an

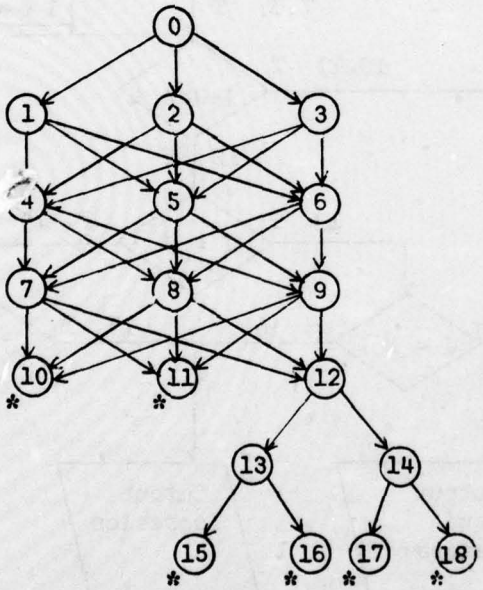



FIGURE 15 -- GRAPH DRAWN FROM THE FLOWCHART IN FIGURE 14.

asterisk, and now there are only six. The comparisons in segments 12 through 18 are simplified by the ordering of the variables so that, for example, I cannot be greater than J, nor J than K, nor can I be equal to K unless it is also equal to J. In an isosceles triangle, J must be one of the equal sides.

A structured program realizing the flowchart of Figure 14 is given in Appendix C. Six cases which test the program are given in Table 4, while the

TABLE 4 -- TEST CASES FOR THE FLOWCHART IN FIGURE 14.

| Test No. | Data | Path | Results |
|---|---|---|---|
| 1 | I=1 J=1 K=1 | 2-5-8-12-13-15 | Equilateral |
| 2 | 2 3 2 | 1-6-8-12-14-16 | Isósceles |
| 3 | 3 2 2 | 3-5-7-12-14-17 | Isosceles |
| 4 | 2 3 4 | 1-4-7-12-14-18 | Scalene |
| 5 | 9 5 4 | 3-6-9-11 | Not a triangle |
| 6 | 10 5 4 | 3-6-9-10 | Not a triangle |

test results are also shown in Appendix C. Section 4 of this report shows how the size and number of the maximum incomparable set can be found in a formal manner.

## 2.4 Program Writing and Debugging Experience.

As measured by the number of test cases required, the unstructured program is more complex (17 cases) than the structured program using essentially the same solution technique (10 cases). An improved solution technique resulted in an even simpler structured program (6 cases).

TABLE 5 -- COMPARISON OF THE THREE DESIGNS FOR THE TRIANGLE PROBLEM.

| Design | Number of Inputs | Number of Outputs | Number of Decisions | Number of Assignments | Number of Incomparable Elements | Number of Tests |
|---|---|---|---|---|---|---|
| 1-(Fig. 9) | 1 | 4 | 13 | 4 | 17 | 17 (Table 2) |
| 2-(Fig. 12) | 1 | 4 | 6 | 4 | 9 | 10 (Table 3) |
| 3-(Fig. 14) | 1 | 4 | 7 | 3 | 6 | 6 (Table 4) |

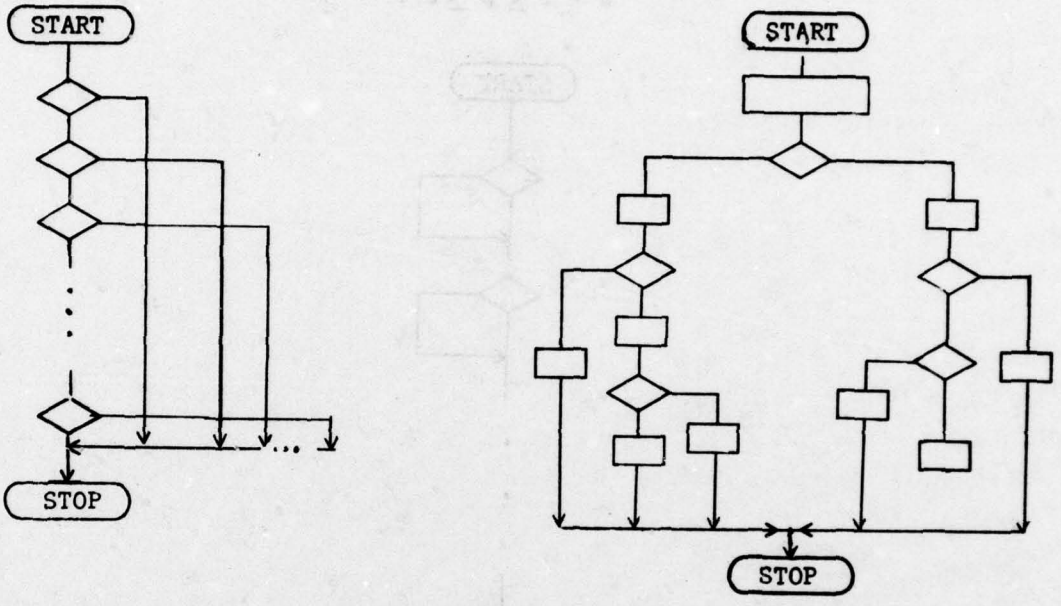
A comparison of the three designs appears in Table 5. Note that all of the designs required the same number of input and output statements. Design 1 requires 17 decision plus assignment statements, while Designs 2 and 3 require only 10. Thus, the second and third designs have 7 fewer statements

each (if programmed in a higher level language), than does Design 1. In terms of the number of type 1.5 level tests, Design 3 requires the fewest. Based on the above analysis, we have not only been able to assign the number of tests required for each design, but also develop information useful as a comparative rating of the designs.

The problem as given was so easy that the debugging of the three programs provided no enlightenment. No logic errors were found in any of the programs, although one PL/C syntax error and one job control error were found in the unstructured program, three input data errors in the first of the structured programs, and one input data error in the second. The reported debugging advantages of structured programming evidently become apparent only in larger programs.

## 3.0 Relations Among Number of Paths and Number of Deciders.

In this section we will discuss some of the relationships between the number of deciders in a loopless flowchart, and the number of paths when the program contains no loops. A loopless flowchart is said to be strictly branching if it contains no merges other than a final merge before the final STOP. The two flowcharts in Figure 16 are strictly branching. Taking




a. b.

FIGURE 16 -- TWO STRICTLY BRANCHING FLOWCHARTS.

all the branches as two-way branches, the number of paths in each flowchart is given by:

$$p = d + 1 \qquad (1)$$

where d is the number of deciders. It can be seen from the flowchart in Figure 16(a) that Eq. (1) always holds. Given any strictly branching flowchart, if any decider were added to the flowchart, exactly one path would be added. If the flowchart contained only one decider there would be two paths, so the difference between p and d is always 1.

If a loopless flowchart contains d deciders, the smallest number of paths it can have is given by Eq. (1), and that value is obtained when there are no merges. If the flowchart contains merges, the number of paths is larger, and the largest number of paths is obtained when each decider is

followed by a merge, as in Figure 17. Here the number of paths is given by:

$$p = 2^d \tag{2}$$

so the number of paths in any loopless flowchart is:

$$d + 1 \leq p \leq 2^d. \tag{3}$$

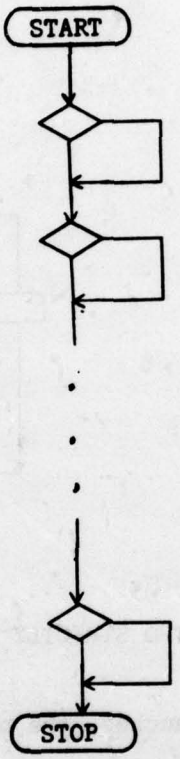



FIGURE 17 -- A FLOWCHART WITH A MERGE FOLLOWING EACH DECIDER.

Let us apply Eq. (3) to Figure 18. Here there are six deciders, so the number of flowchart paths as given by Eq. (3) is:

$$7 \leq p \leq 64.$$

There are in fact 27 paths in the flowchart, and a closer lower bound can be obtained than that given by Eq. (3) by decomposition of the flowchart at convenient locations. Figure 19 shows the flowchart divided at the merge points. Now bounds for each section of the flowchart may be

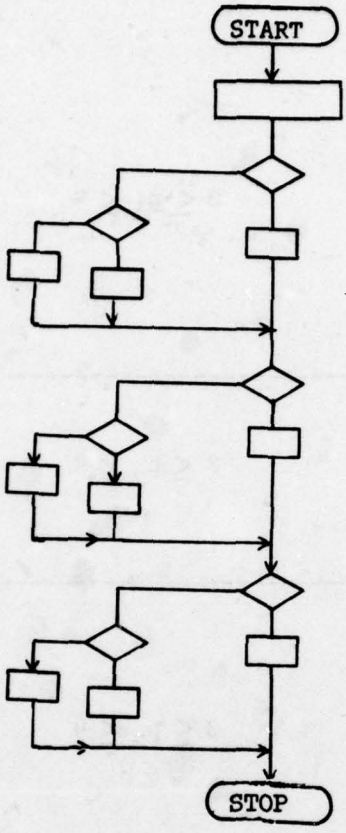



FIGURE 18 -- A FLOWCHART WITH SIX DECIDERS AND THREE MERGES.

computed using Eq. (3), and we find:

$$3 \leq p_1, p_2, p_3 \leq 4$$

since each section of the flowchart contains 2 deciders. The bounds on the complete flowchart are the product of the individual bounds, or:

$$27 \leq p \leq 64.$$

Note that the upper bound is unchanged.

The flowchart appearing in Figure 5 may be similarly decomposed at a single merge point. Figure 20 shows the decomposition of the original 13 deciders into groups of 3 and 10. This yields:

$$4 \leq p_1 \leq 2^3$$

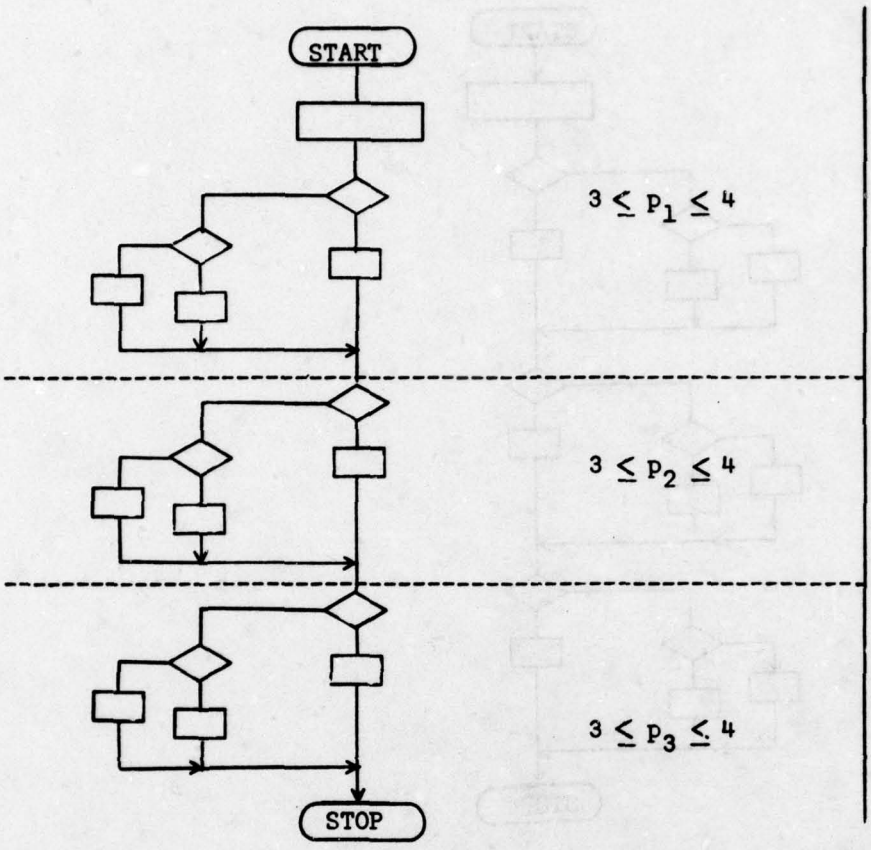



FIGURE 19 -- FIGURE 18 DIVIDED AT THE MERGE POINTS.

$$11 \leq p_2 \leq 2^{10}$$

and, multiplying:

$$44 \leq p \leq 2^{13}.$$

The actual number of paths in Figure 5 is 88.

The flowchart appearing in Figure 14 may be decomposed at segment 7, 8, 9. The decomposition is not shown, but the original 7 deciders divide into groups of 3 and 4. The bounds from the decomposition are:

$$4 \leq p_1 \leq 8$$

$$5 \leq p_2 \leq 16$$

$$20 \leq p \leq 128$$

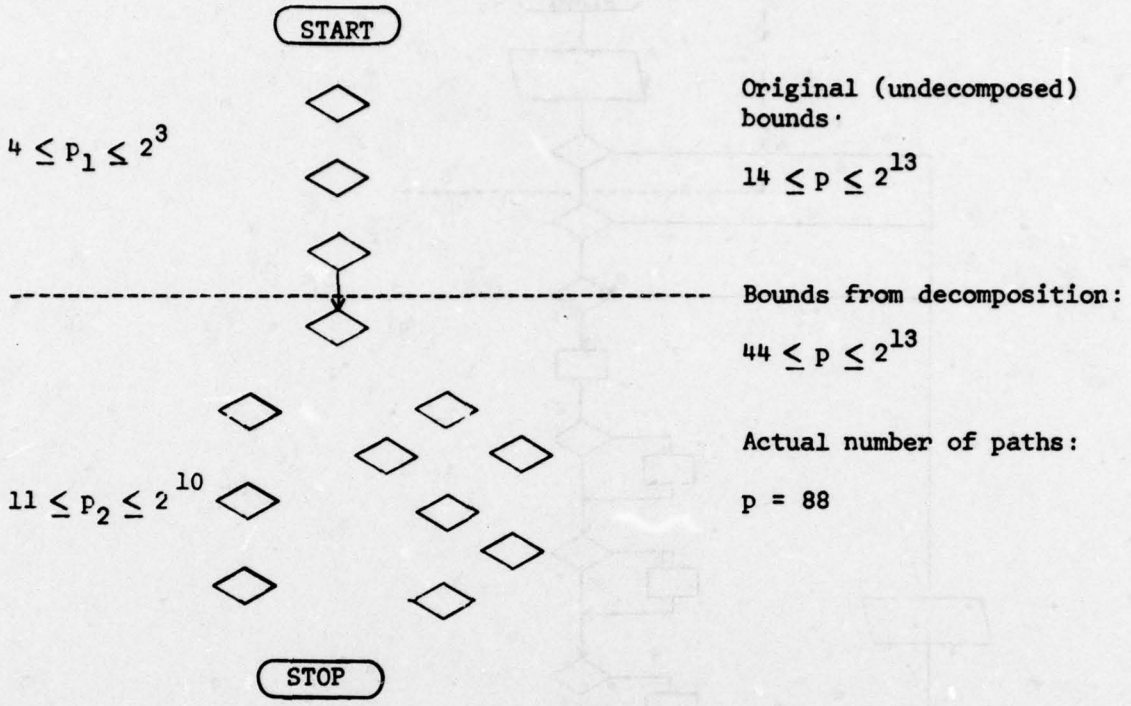



FIGURE 20 -- FIGURE 5 DIVIDED AT ONE MERGE POINT.

which compares to the original bounds of $8 \leq p \leq 128$. The actual number of paths is 40.

In the three examples so far, each merge point was a "cut set" of the flowchart, where the cutting of the flow at the merge point separated the START from the STOP. In each of the three cases the lower bound was improved (raised) by decomposition but, for obvious reasons, the upper bound was not affected. In fact, the lower bound after decomposition became a good approximation. A different situation is obtained with the flowchart in Figure 12. Now it is not possible to find a single point which constitutes a cut set (except trivial ones at the very beginning and end), and it is not obvious how the flowchart can best be decomposed. The six deciders can be divided into 1 and 5, or 2 and 4, etc., (if we limit ourselves to a two-piece decomposition), and each decomposition gives a different upper bound, although the lower bound remains 7 in all of the cases. Clearly, that decomposition which gives the minimum upper bound is the best, and it is shown in Figure 21. Figure 22 summarizes Figure 21, and the numbers in the boxes of Figure 22 represent the number of deciders in each part of the original flowchart. Since the cut set consists of two segments, the number of paths through the flowchart will be the sum of the numbers of paths through each of the segments. Let $p_1'$ be the number of $p_1$ paths

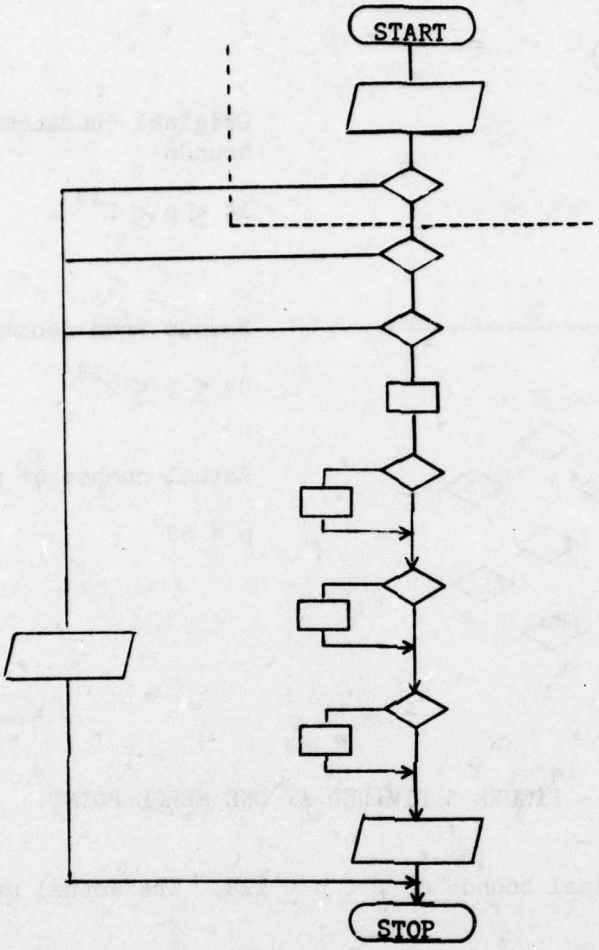



FIGURE 21 -- THE "BEST" WAY TO DECOMPOSE FIGURE 12.

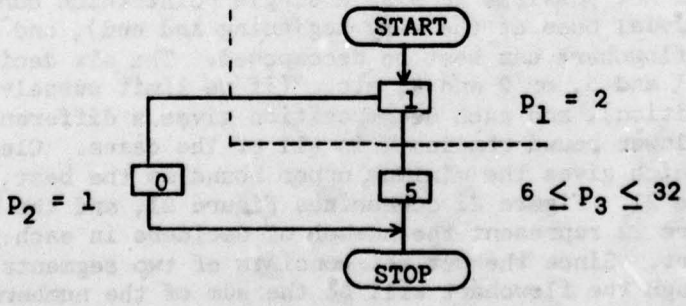



FIGURE 22 -- FIGURE 21 SUMMARIZED.

that pass through $p_2$; let $p_1''$ be the number of $p_1$ paths that pass through $p_3$. Then the number of paths through the flowchart is given by:

$$p = p_1' p_2 + p_1'' p_3. \tag{4}$$

In this case, $p_1' = p_1'' = p_2 = 1$, and $6 \leq p_3 \leq 32$, so that:

$$7 \leq p \leq 33,$$

compared to the original bounds of $7 \leq p \leq 64$ without decomposition. The actual number of paths through the flowchart is 11. Other two-piece decompositions give higher upper bounds.

It is possible to obtain even lower upper bounds for this flowchart by using three- and four-piece decompositions. But a four-piece decomposition is nearly a block-by-block analysis of the flowchart, and at that point one can compute the number of paths directly as in [2]. Additional work is required in this area to determine how to most effectively decompose flowcharts.

Since decomposition of a flowchart at other than a merge point lowers the upper bound, we can return to the flowcharts shown in Figures 18 and 20 and decompose them at other than merge points. They were previously decomposed at merge points only, thereby raising their lower bounds. A three-piece decomposition of Figure 18 is shown in Figure 23, and summarized in Figure 24. The numbers shown in the boxes of Figure 24 show the number of deciders in each portion of the decomposition. Let $p_1'$ be the number of $p_1$ paths that pass through $p_2$; let $p_1''$ be the number of $p_1$ paths that go directly to $p_3$. Then the number of paths through the flowchart is given by:

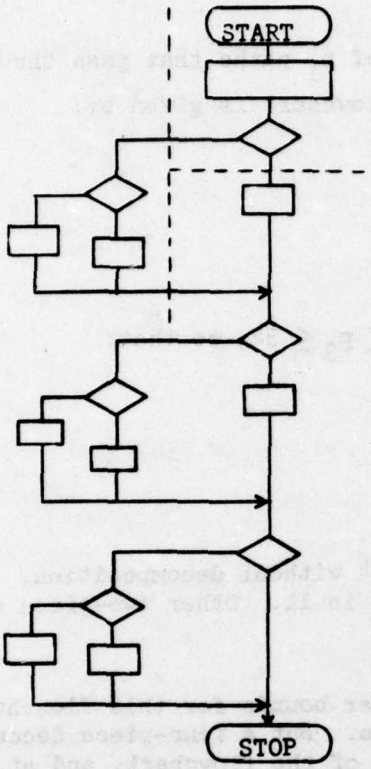



FIGURE 23 -- THREE-PIECE DECOMPOSITION OF THE FLOWCHART IN FIGURE 18 AT OTHER THAN A MERGE POINT.

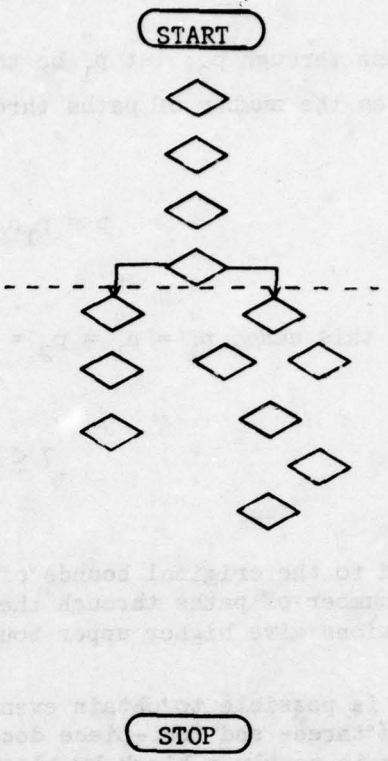



FIGURE 25 -- DECOMPOSITION OF THE FLOWCHART IN FIGURE 20 AT OTHER THAN A MERGE POINT.

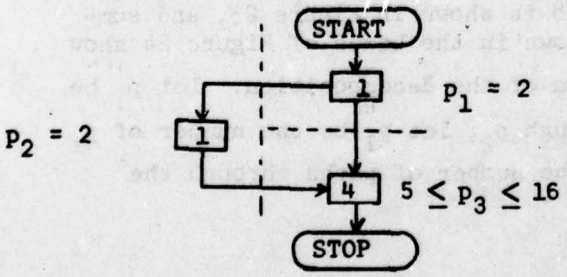



FIGURE 24 -- FIGURE 23 SUMMARIZED.

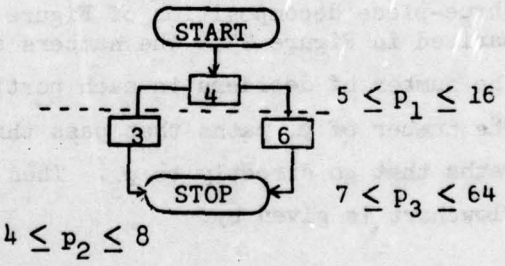



FIGURE 26 -- FIGURE 25 SUMMARIZED.

$$p = p_1' p_2 p_3 + p_1'' p_3. \qquad (5)$$

In this case $p_1' = p_1'' = 1$, $p_2 = 2$, and $5 \leq p_3 \leq 16$, so that:

$$15 \leq p \leq 48.$$

This compares to the original undecomposed bounds of $7 \leq p \leq 64$, and the bounds after decomposition at a merge point of $27 \leq p \leq 64$. Combining the results of both decompositions gives bounds of $27 \leq p \leq 48$.

The flowchart whose outline is shown in Figure 20 may be decomposed at other than a merge point to reduce the upper bound dramatically from its undecomposed value of $2^{13}$. The decomposition is shown in Figure 25 and summarized in Figure 26. The number of paths through the flowchart is given by:

$$p = p_1 p_2 + p_1 p_3. \qquad (6)$$

In this case, $5 \leq p_1 \leq 16$, $4 \leq p_2 \leq 8$, and $7 \leq p_3 \leq 64$, so that:

$$55 \leq p \leq 1152.$$

## 4.0 Matrices and Zero-One Integer Linear Programming

Lipow describes the properties of adjacency matrices that can be constructed from the graphs of Section 2 of this report (which he incorrectly calls incidence matrices), and shows how matrix manipulation will yield information about the total number of paths in the flowchart. He does not show how the matrices may be used to find the size and members of the maximum incomparable set. That will be done in this section.

First, the construction and manipulation of the matrices, as described by Lipow, will be summarized. Then, using his example, it will be shown how a zero-one integer linear programming problem can be formulated in a completely mechanical way from the matrix, whose solution yields the size and members of the maximum incomparable set. Finally, another example, a flowchart with loops, will be examined.

### 4.1 Matrices

For the purposes of formulating a linear programming problem to find the maximum incomparable set for a flowchart, it is necessary to develop a single matrix which describes, in some way, all of the connecting flows that can be found in the flowchart. We do this by first constructing an adjacency matrix from the graph of a flowchart, in which the rows represent predecessor segments and the columns represent successor segments. The matrix contains a 1 wherever a predecessor segment immediately precedes a successor segment, and contains a 0 everywhere else. Such a matrix can be constructed in a finite number of steps by first considering segment 1 as the predecessor, and filling in a 1 for each of its successors; then considering segment 2 as the predecessor and filling in a 1 for each of its successors. No row need be constructed for segment 0 since it can never be a member of the maximum incomparable set. Thus the matrix will be constructed row by row. The adjacency matrix, M. constructed from the graph appearing in Figure 6 is shown in Figure 27. Note that the elements having rows with all zeros constitute an incomparable set. This may not be the maximum incomparable set, however, as will be seen in a later example.

From the matrix M, compute $M^2$, $M^3$, etc. The matrix $M^r$ represents the matrix M multiplied by itself r-1 times and the numerical values of its elements represent the number of paths by which a successor segment can be reached from a predecessor segment in exactly r steps. The set of matrices $M, M^2, M^3, \ldots, M^k$ exhibits all segments that can be reached from any other segment in the flowchart. If there are no loops in the flowchart, then k is such that:

$$M^{k+1} = 0, \tag{7}$$

SUCCESSORS

| PREDECESSORS | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

FIGURE 27 -- ADJACENCY MATRIX FOR THE GRAPH OF FIGURE 6.

the zero matrix. If the flowchart contains loops, however, the value of k may not be easy to find, but it cannot be larger than the number of segments in the flowchart. The number of segments thus serves as an upper bound on k in a looping flowchart. It is sometimes possible to find k in such a flowchart, and this will be done later in this section when the looping flowchart is discussed.

To obtain a single matrix exhibiting all segment relationships needed, we may sum the matrices in the set (by adding corresponding elements) to obtain the matrix T:

$$T = \sum_{r=1}^{k} M^r. \tag{8}$$

This has been done for the matrix of Figure 27, and the resulting sum is shown in Figure 28 (c.f. Lipow, op.cit.). We may now construct, from the matrix T, a zero-one integer linear programming problem whose solution is the size and members of the maximum incomparable set. The elements with all zero rows in T are the same as the elements with all zero rows in M, and those elements constitute an incomparable set, but not necessarily the maximum incomparable set.

## 4.2 Zero-One Integer Linear Programming

To construct the zero-one integer linear programming problem (binary programming problem) whose solution yields the size and members of the maximum incomparable set, first form the matrix F from the matrix T as follows:

(a) Set all the diagonal elements of F equal to the number of flowchart segments as:

$$f_{ii} = s \qquad i = 1, \ldots, s$$

(b) Set the elements above the diagonal of F equal to 0 or 1, based upon the elements of the T matrix according to the following formula:

$$f_{ij} = \begin{cases} 1 \text{ if } t_{ij} + t_{ji} \neq 0, \; i < j \\ 0 \text{ if } t_{ij} + t_{ji} = 0, \; i < j \end{cases}$$

(c) Set all elements below the diagonal of F equal to zero:

$$f_{ij} = 0, \quad i > j$$

T =

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 2 | 2 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 2 | 0 | 0 | 1 | 1 | 2 | 2 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 3 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 4 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**FIGURE 28 -- THE MATRIX T FOR THE MATRIX M OF FIGURE 27.**

The binary programming problem will use variables $x_i$, $i = 1,\ldots, s$ and in the solution:

$$x_i = \begin{cases} 1 \text{ if segment } i \text{ is a member of the maximum incomparable set} \\ 0 \text{ if segment } i \text{ is not a member of the maximum incomparable set.} \end{cases}$$

The binary programming problem is:

$$\text{maximize} \sum_{i=1}^{s} x_i$$

subject to:

$$FX \leq S$$

where X is a column vector and $X' = (x_1, x_2,\ldots, x_s)$, and S is a column vector of order s, and $S' = (s, s,\ldots, s)$. In the solution, the value of the objective function gives the size of the maximum incomparable set, and the members of the set are indicated by the $x_i = 1$.

As an example, forming F from the matrix T in Figure 28 yields the matrix shown in Figure 29. The binary programming problem is then:

$$\text{maximize} \sum_{i=1}^{26} x_i$$

subject to the constraints given in Table 6.

The solution to the binary programming problem has an objective value of 11, and:

$$x_{11}, x_{15}, x_{16}, x_{17}, x_{19}, x_{21}, x_{22}, x_{23}, x_{24}, x_{25}, x_{26} = 1$$

$$\text{all other } x_i = 0,$$

giving the size and members of the maximum incomparable set. In this case, by chance, the maximum incomparable set happens to consist of the elements having all zero rows in the M and T matrices. That this need not be so is shown in the next example.

$$F = \begin{array}{c|cccccccccccccccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 \\
\hline
1 & 26 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
2 & 0 & 26 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
3 & 0 & 0 & 26 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
4 & 0 & 0 & 0 & 26 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
5 & 0 & 0 & 0 & 0 & 26 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
6 & 0 & 0 & 0 & 0 & 0 & 26 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
14 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
17 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 1 & 1 \\
21 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 & 0 \\
22 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 & 0 \\
23 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 & 0 \\
24 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 & 0 \\
25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 0 \\
26 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26
\end{array}$$

FIGURE 29 — THE MATRIX F FORMED FROM T IN FIGURE 28.

TABLE 6 — CONSTRAINTS FOR THE BINARY PROGRAMMING PROBLEM.

$$
\begin{array}{l}
26x_1 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17} + x_{18} + x_{19} + x_{20} + x_{21} + x_{22} + x_{23} + x_{24} + x_{25} + x_{26} \leq 26 \\
26x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17} + x_{18} + x_{19} + x_{20} + x_{21} + x_{22} + x_{23} + x_{24} + x_{25} + x_{26} \leq 26 \\
26x_3 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17} + x_{18} + x_{19} + x_{20} + x_{21} + x_{22} + x_{23} + x_{24} + x_{25} + x_{26} \leq 26 \\
26x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17} + x_{18} + x_{19} + x_{20} + x_{21} + x_{22} + x_{23} + x_{24} + x_{25} + x_{26} \leq 26 \\
26x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17} + x_{18} + x_{19} + x_{20} + x_{21} + x_{22} + x_{23} + x_{24} + x_{25} + x_{26} \leq 26 \\
26x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17} + x_{18} + x_{19} + x_{20} + x_{21} + x_{22} + x_{23} + x_{24} + x_{25} + x_{26} \leq 26 \\
26x_7 + x_9 + x_{10} + x_{13} + x_{14} + x_{15} + x_{16} + x_{19} + x_{20} + x_{21} + x_{22} + x_{25} + x_{26} \leq 26 \\
26x_8 + x_{11} + x_{12} + x_{17} + x_{18} + x_{23} + x_{24} \leq 26 \\
26x_9 + x_{13} + x_{14} + x_{19} + x_{20} + x_{21} + x_{22} + x_{25} + x_{26} \leq 26 \\
26x_{10} + x_{15} + x_{16} \leq 26 \\
26x_{11} \leq 26 \\
26x_{12} + x_{17} + x_{18} + x_{23} + x_{24} \leq 26 \\
26x_{13} + x_{19} + x_{20} + x_{25} + x_{26} \leq 26 \\
26x_{14} + x_{21} + x_{22} \leq 26 \\
26x_{15} \leq 26 \\
26x_{16} \leq 26 \\
26x_{17} + x_{23} + x_{24} \leq 26 \\
26x_{18} \leq 26 \\
26x_{19} + x_{25} + x_{26} \leq 26 \\
26x_{20} \leq 26 \\
26x_{21} \leq 26 \\
26x_{22} \leq 26 \\
26x_{23} \leq 26 \\
26x_{24} \leq 26 \\
26x_{25} \leq 26 \\
26x_{26} \leq 26
\end{array}
$$

$$x_i = 1,0; \quad i = 1, \ldots, 26$$

## 4.3 A Looping Flowchart

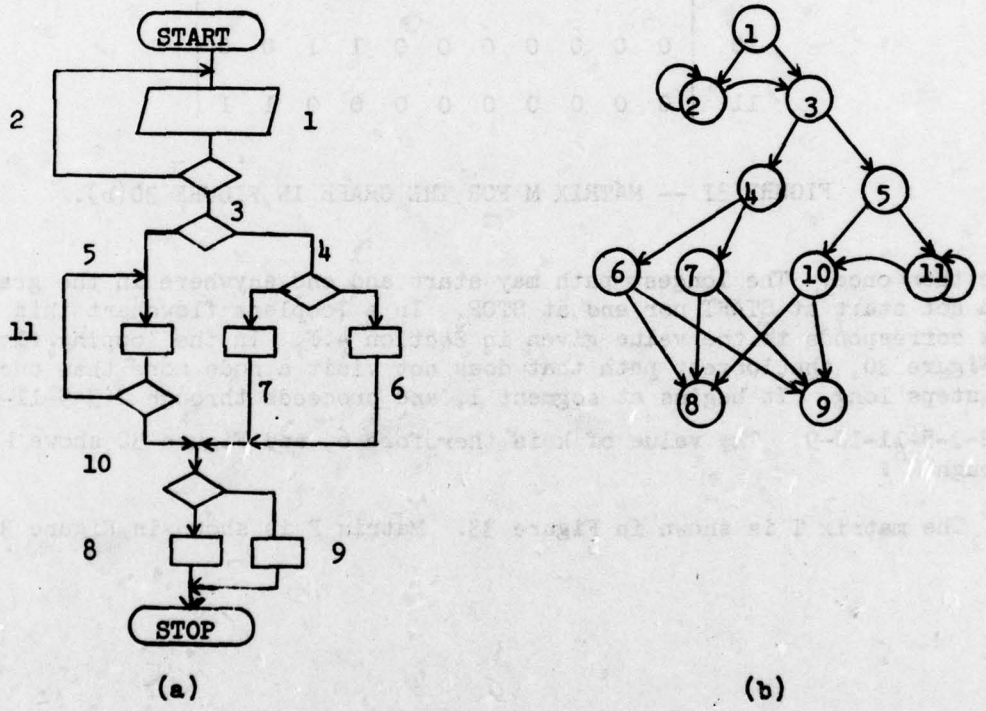
The example used in Section 4.2 is less general than we would like in several respects:

a) the members of the maximum incomparable set all happen to be flowchart terminators, appear at the bottom of the graph, and have all zero rows in M and T,

b) the segments are numbered in such a way that the lower left portion of the T matrix happens to contain all zeros ($t_{ij} = 0$, all $i > j$),

c) the flowchart contains no loops.

A simple flowchart which does not contain these special conditions is found in Figure 30(a), and the graph of the flowchart in Figure 30(b). The




(a) (b)

FIGURE 30 -- A FLOWCHART AND ITS GRAPH

matrix M is given in Figure 31. The higher powers of M are given in Figure 32. The upper bound on k, the number of segments, is 11, but in this case the flowchart is simple enough that the exact value of k can be found. In any flowchart, looping or otherwise, k is equal to the number of steps in the longest path that can be traversed in the graph without visiting a node

| M = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

FIGURE 31 -- MATRIX M FOR THE GRAPH IN FIGURE 30(b).

more than once. The longest path may start and end anywhere in the graph; it need not start at START nor end at STOP. In a loopless flowchart this value of k corresponds to the value given in Section 4.1. In the looping flowchart of Figure 30, the longest path that does not visit a node more than once is six steps long. It begins at segment 1, and proceeds through 2-3-5-11-10-8; or 2-3-5-11-10-9. The value of k is therefore 6, and Figure 32 shows $M^2$ through $M^6$.

The matrix T is shown in Figure 33. Matrix F is shown in Figure 34.

$M^2$ =

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

$M^3$ =

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 1 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

$M^4$ =

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 3 | 2 | 2 |
| 2 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 3 | 2 | 2 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

$M^5$ =

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 4 | 4 | 3 | 3 |
| 2 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 4 | 4 | 3 | 3 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

$M^6$ =

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 5 | 5 | 4 | 4 |
| 2 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 5 | 5 | 4 | 4 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

FIGURE 32 -- COMPUTATION OF MATRICES $M^2$ THROUGH $M^6$.

$T =$

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

FIGURE 33 -- T FOR THE SET OF MATRICES OF FIGURES 31 and 32.

$F =$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 11 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 11 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 0 | 0 | 11 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 0 | 0 | 0 | 11 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 11 | 0 | 0 | 1 | 1 | 1 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 11 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 1 | 1 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 0 | 1 | 1 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 1 | 1 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 1 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 11 |

FIGURE 34 -- MATRIX F FROM THE MATRIX T OF FIGURE 33.

The binary programming problem is:

$$\text{maximize} \quad \sum_{i=1}^{11} x_i$$

subject to:

$$
\begin{aligned}
11x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} &\le 11 \\
11x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} &\le 11 \\
11x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} &\le 11 \\
11x_4 + x_6 + x_7 + x_8 + x_9 &\le 11 \\
11x_5 + x_8 + x_9 + x_{10} + x_{11} &\le 11 \\
11x_6 + x_8 + x_9 &\le 11 \\
11x_7 + x_8 + x_9 &\le 11 \\
11x_8 + x_{10} + x_{11} &\le 11 \\
11x_9 + x_{10} + x_{11} &\le 11 \\
11x_{10} + x_{11} &\le 11 \\
11x_{11} &\le 11
\end{aligned}
$$

$$x_i = 1,0; \quad i = 1,\ldots, 11$$

The solution has an objective value of three, and:

$$x_i = 1, \begin{Bmatrix} i = 5,6,7 \\ i = 6,7,10 \\ i = 6,7,11 \end{Bmatrix} \text{ all other } x_i = 0$$

The objective value of three indicates that the maximum incomparable set contains three elements, and hence that three tests are needed to execute every segment of the flowchart at least once. The variables $x_i$ indicate which segments must be tested in the test cases. Here there are three optimal solutions to the binary programming problem, and any one of the three could be used with equivalent results. In the first solution, the $x_i$ indicate that segments 5, 6, and 7 must be each executed once and only once in the three test cases. Equivalently, the other two solutions indicate that segments 6, 7, and 10 must be tested once and only once; or that segments 6, 7, and 11 must be so tested.

## 4.4 Matrix Reductions

Sometimes the matrix F can be reduced by elimination of rows or rows and columns, and sometimes some of the variables can be removed

from the vector X and from the objective function and set equal to 0 or 1 before solution of the binary programming problem is begun.

If, for any variable $x_r$, all the elements in column r of F above the main diagonal are equal to 1, and all the elements in row r of F to the right of the main diagonal are equal to 1, that is, if:

$$f_{ir} = 1 \text{ for all } i < r, \text{ and}$$
$$f_{rj} = 1 \text{ for all } j > r \tag{9}$$

then remove $x_r$ from X and from the objective function, set $x_r = 0$, and remove row r and column r from F, unless reduction (9) holds for all r. If reduction (9) holds for all r, then the binary programming problem is a trivial one, with an objective value of one, and with any one variable comprising the maximum incomparable set.

If, for any variable $x_r$, all the elements of column r and row r of F above and to the right of the main diagonal are zero, that is, if:

$$f_{ir} = 0 \text{ for all } i < r, \text{ and}$$
$$f_{rj} = 0 \text{ for all } j > r \tag{10}$$

then remove $x_r$ from X and from the objective function, set $x_r = 1$, and remove row r and column r from F. Also check for a flowcharting or programming error, for this condition indicates that segment r cannot reach any other segment nor is it reachable from any other segment.

If, for any variable $x_r$, column r of F contains at least one non-zero element above the main diagonal and all the elements of row r of F to the right of the main diagonal are zero, that is, if:

$$f_{ir} = 1 \text{ for any } i, \text{ and}$$
$$f_{rj} = 0 \text{ for all } j > r \tag{11}$$

then remove row r from F. Leave $x_r$ in X and in the objective function, and leave column r in F.

These reductions will now be applied to the matrices F in Figure 29 and Figure 34.

In Figure 29, reduction (11) permits the removal of rows 15, 16, 17, 19, 21, 22, 23, 24, 25, and 26 from F.

In Figure 34, reduction (9) permits the removal of rows 1, 2, and 3, and columns 1, 2, and 3 from F, along with the removal of $x_1$, $x_2$, and $x_3$ from X and from the objective function, and the setting of $x_1 = x_2 = x_3 = 0$. Reduction (11) permits the removal of row 11 from F. The reduced F from Figure 34 is shown in Figure 35.

The solution is the same as before the reduction.

| | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 11 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | | $x_4$ | | | 11 |
| 5 | 0 | 11 | 0 | 0 | 1 | 1 | 1 | 1 | | $x_5$ | | | 11 |
| 6 | 0 | 0 | 11 | 0 | 1 | 1 | 0 | 0 | | $x_6$ | | | 11 |
| 7 | 0 | 0 | 0 | 11 | 1 | 1 | 0 | 0 | x | $x_7$ | $\leq$ | | 11 |
| 8 | 0 | 0 | 0 | 0 | 11 | 0 | 1 | 1 | | $x_8$ | | | 11 |
| 9 | 0 | 0 | 0 | 0 | 0 | 11 | 1 | 1 | | $x_9$ | | | 11 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 1 | | $x_{10}$ | | | 11 |
| | | | | | | | | | | $x_{11}$ | | | |

FIGURE 35 -- THE REDUCED MATRIX FROM FIGURE 34.

## 4.5 Additional Comments on the Matrices

The matrices M, $M^2$,..., $M^k$ by themselves yield information about the flowchart and graph, aside from their usefulness in forming T and the linear programming problem.

For example, in any matrix where:

$$m_{ii}^{(r)} \neq 0$$

a loop is indicated. If r = 1, then the loop is a "self-loop," or a one-step loop, where the flow returns to the same decider in one step. If r > 1, then r is the number of segments in the loop. In Figure 31, $m_{22} = 1$, indicating a self-loop in node 2.

Any:

$$m_{ij}^{(r)} > 1 \qquad i \neq j$$

indicate parallel paths between segment i and segment j. Parallel paths cannot occur with r = 1, but for matrices of higher powers parallel paths are frequently found. For example, in Figure 32 $m_{2,9}^{(4)} = 3$, indicating that there are three different ways to go from segment 2 to segment 9 in four steps. Different numbers of times around a loop are considered a different way to go, and the three ways are 2-3-4-6-9, 2-3-4-7-9, and 2-3-5-10-9.

## 4.6 Upper and Lower Bounds on the Number of Tests Needed

Using the flowchart in Figure 36 (decisions tested only two ways, not three) with its segments numbered, and following the methods in Section 2 and Sections 4.1 through 4.3, the matrices M and T may be constructed with k = 7 (Table 4). The methods of those sections also yield 5 as the size of the maximum incomparable set, which is the lower bound on the minimum number of tests needed to pass through every segment at least once.

The actual minimum number of test cases required will depend on the contents of the flowchart boxes. It is conjectured that sufficient conditions for achieving the lower bound are:

a) the decisions are all independent of one another,

b) the decision variables are all read as input,

c) the program does not modify the values of the decision variables.

If, on the other hand, the decisions have a certain degree of dependence, the lower bound may not be achievable (as in the flowchart in Figure 12). The flowchart in Figure 37, for example, has the same structure as that in Figure 36, but cannot be tested with only five cases. Owing to the dependence of the decisions, six tests are needed in order to pass through every segment at least once. There are several different sets of six tests which will exercise every segment, but no set smaller than 6.

By changing the contents of the boxes in the flowchart in Figure 37, it is possible to create a different flowchart (but with the same structure) that will require seven tests to exercise all segments.

In the case of a flowchart without loops, the upper bound on the minimum number of test cases needed to pass through every segment at least once, u, is given by:

$$u = d + 1$$

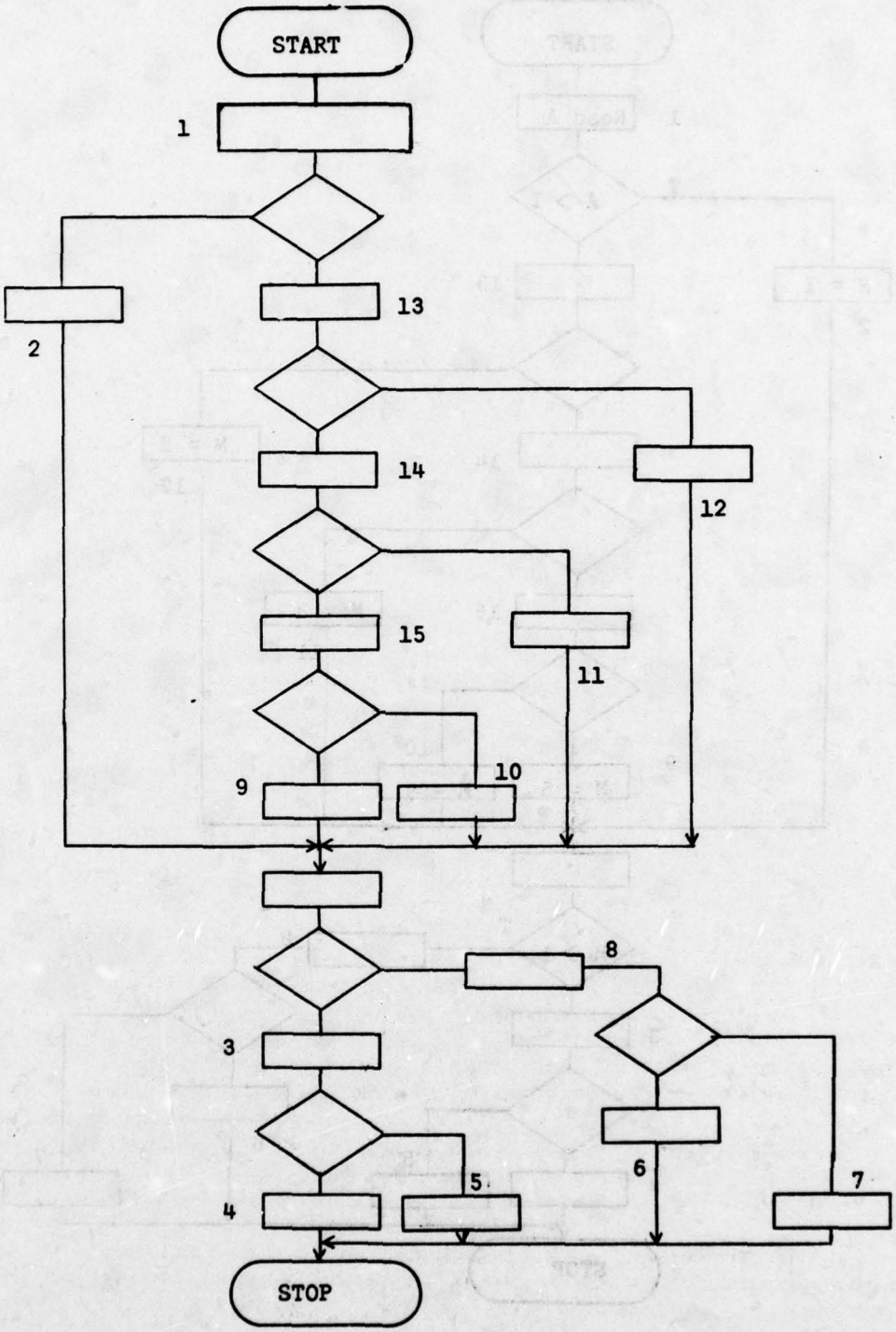



FIGURE 36 -- A FLOWCHART.

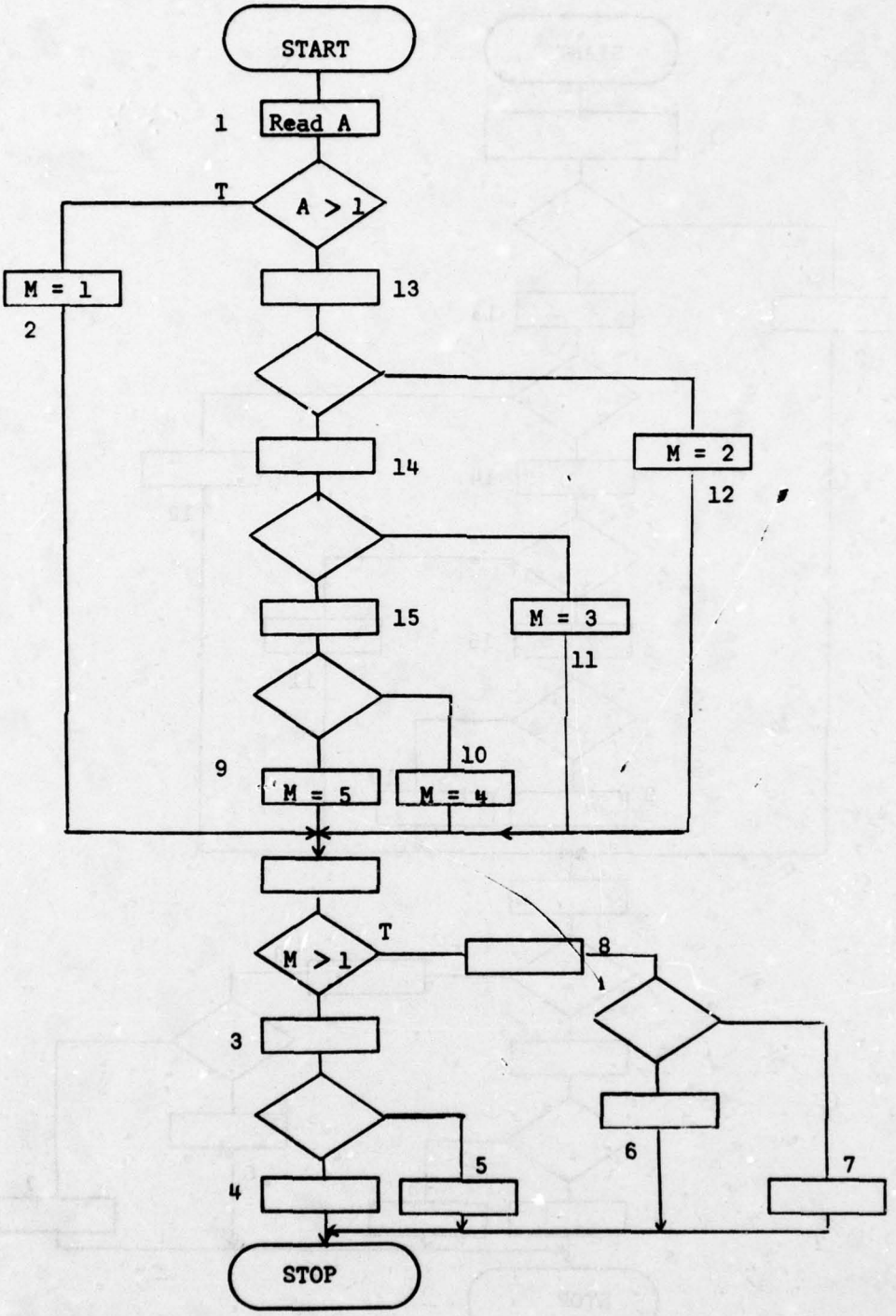



FIGURE 37 -- THE FLOWCHART OF FIGURE 36 WITH CONTENTS.

where d is the number of deciders in the flowchart. (This is the same formula as the one for determining the smallest number of paths that a flowchart may have, given the number of deciders in the flowchart, as given in Section 3.0.)

Proof: Consider a flowchart with no deciders. Such a flowchart has one segment and requires one test. Each decider added to the flowchart can require at most one additional test, so $u = d + 1$. (This proof bears some resemblance to the derivation of the formula in Section 3.0, although there is no obvious connection between the two.) QED

## 4.7 Calculating the Number of Paths and Enumerating the Paths in a Flowchart, using Matrices

The matrix T in Table 4 can be used to readily reveal the total number of paths in the flowchart of Figure 36, as shown by Lipow in [1]. Since all paths begin at segment 1, and end either at segment 4, 5, 6, or 7, the total number of paths, p, is

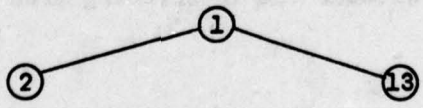
$$p = t_{1,4} + t_{1,5} + t_{1,6} + t_{1,7}.$$

This is seen to be 20, a number which can be obtained also by the decomposition method described in Section 3 or by the direct calculation method described by Shooman in [2].

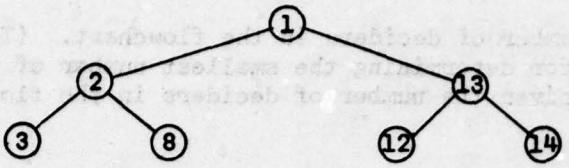
An enumeration of the 20 paths may be generated from the matrix M, and the enumeration takes the form of a tree.

Since all 20 paths begin with segment 1, we start the tree with node 1.


From the matrix M, row 1 indicates that segments 2 and 13 can follow segment 1, so 2 and 13 are added to the tree.




Row 2 of M indicates that 3 and 8 follow 2, and row 13 indicates that 12 and 14 follow 13. These are now added to the tree.




The tree is built level by level, with each bottom terminus of the tree directing to a row of M, until construction of the tree terminates due to a row of M containing all zeros. This process was applied to the flowchart of Figure 36, and the result is shown in Figure 38, showing the segments that comprise each of the 20 paths in the flowchart.

The paths so enumerated can now be represented by a matrix, which will prove useful in Section 4.8. In the matrix, P, each column represents a path, and contains a 1 for each segment in the path. The matrix P from the tree in Figure 38 is given in Figure 39.

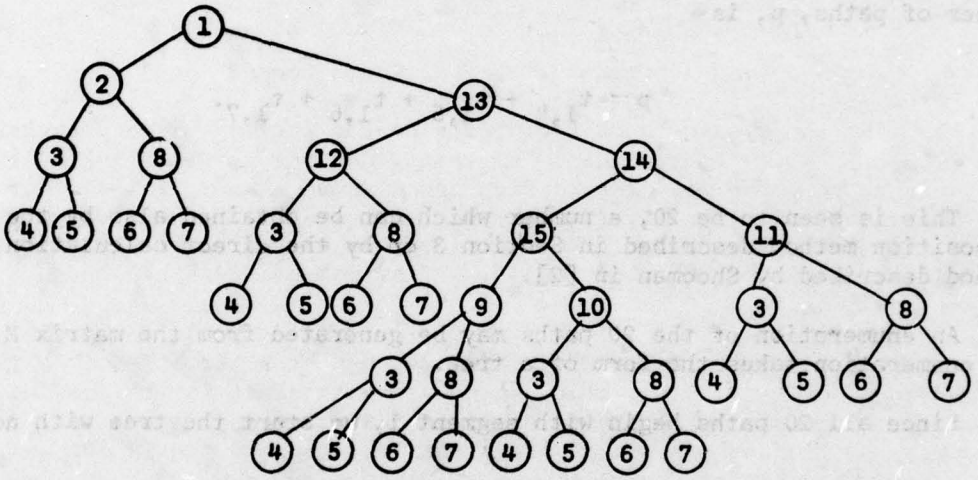



FIGURE 38 -- A TREE SHOWING ALL THE PATHS IN THE FLOWCHART OF FIGURE 37.

Further work is needed to develop a method of generating P from M by matrix operations, without the intervening tree enumeration.

$$
P = \begin{array}{c|cccccccccccccccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
2 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
5 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
7 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
8 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
12 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
13 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
14 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0
\end{array}
$$

FIGURE 39 -- THE PATH MATRIX FOR THE PATH TREE IN FIGURE 38.

## 4.8 Finding the Actual Minimum Number of Tests Needed

To find the minimum number of tests needed for any given flowchart, start with the path matrix P and remove any columns which represent infeasible paths. For example, to find the minimum number of tests for the flowchart in Figure 37, remove from P (Figure 39) columns which represent paths that cannot be traversed. One such path, for example, consists of segements 1, 2, 8, and 7. It cannot be traversed because of the way M is set.

The flowchart in Figure 37 is small enough that the infeasible paths can be found by inspection. For larger flowcharts, it may be possible to use logical operations (AND, OR, etc.) to detect infeasible paths. So far, work on the use of logical operations to detect infeasible paths has not yielded satisfactory results. See [3].

In very large flowcharts where flow relationships are less obvious, it may not be possible to find and remove all infeasible columns from P. Such a defect is not fatal, however, and can be remedied by a procedure to be explained later.

After the infeasible columns are removed, the remaining matrix, U, can be used to construct a binary programming problem whose solution is the number of test cases required, and the paths that those test cases should traverse. Let J be the index set of the feasible paths. Then the binary programming problem is:

$$\text{minimize } z = \sum_{j \in J} x_i$$

subject to:

$$UX \geq 1 \qquad x_j = 0,1$$

where X is a vector whose transpose is $(x_j \mid j \in J)$. The solution yields a value for z which is the actual minimum number of tests needed to pass through all segments at least once. The variables $x_j$ refer to the j feasible paths, and in the solution to the binary programming problem:

$$x_j = \begin{cases} 1, \text{ if path } j \text{ is one of the paths to be traversed by a test} \\ 0, \text{ otherwise} \end{cases}$$

As an example, the 10 feasible paths from P in Figure 39 are 1, 2, 7, 8, 11, 12, 15, 16, 19, and 20. So the binary programming problem is:

$$\text{minimize } z = \sum x_j \quad j = 1,2,7,8,11,12,15,16,19,20$$

subject to:

| | 1 | 2 | 7 | 8 | 11 | 12 | 15 | 16 | 19 | 20 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | $x_1$ | | 1 |
| 2 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | $x_2$ | | 1 |
| 3 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | $x_7$ | | 1 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | $x_8$ | | 1 |
| 5 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | $x_{11}$ | | 1 |
| 6 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | | $x_{12}$ | | 1 |
| 7 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | $x_{15}$ | | 1 |
| 8 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | x | | $x_{16}$ | $\leq$ | 1 |
| 9 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | | | $x_{19}$ | | 1 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | | | $x_{20}$ | | 1 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | | | | | 1 |
| 12 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | 1 |
| 13 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | | 1 |
| 14 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | | | | | 1 |
| 15 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | | | | | 1 |

$$x_j = 0,1; \quad j \in J$$

This is a standard form set covering problem, and the matrix reductions given by Garfinkel and Nemhauser [4] may be applied to U. By their Reduction 4, $r_1 > r_2 = r_3 > r_4$, so rows 1, 2, and 3 are eliminated. Also by 4, $r_8 = r_{13} > r_{14} > r_{15} > r_9$, so rows 8, 13, 14, and 15 are deleted. By reduction 2, $r_4 = e_1$, so row 4 and column 1 are deleted, and $x_1 = 1$. Also by 2, $r_5 = e_2$, so row 5 and column 2 are deleted, and $x_2 = 1$. The problem that remains is:

$$\text{Minimize } z' = \sum x_j \qquad j = 7,8,11,12,15,16,19,20$$

subject to:

$$\begin{array}{c|cccccccc|c|c|c}
 & 7 & 8 & 11 & 12 & 15 & 16 & 19 & 20 & & & \\
\hline
6 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & x_7 & & 1 \\
7 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & x_8 & & 1 \\
9 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & x_{11} & & 1 \\
10 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & x_{12} & \leq & 1 \\
11 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & x_{15} & & 1 \\
12 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & x_{16} & & 1 \\
 & & & & & & & & & x_{19} & & \\
 & & & & & & & & & x_{20} & & 1
\end{array}$$

$$x_j = 0,1; \quad j \in J$$

There are 14 optimal solutions, with $z' = 4$. Some of them are:

$x_7=x_{11}=x_{15}=x_{20}=1$ all other $x_j=0$

$x_7=x_{11}=x_{16}=x_{19}=1$ all other $x_j=0$

$x_7=x_{11}=x_{16}=x_{20}=1$ all other $x_j=0$

$x_7=x_{12}=x_{15}=x_{19}=1$ all other $x_j=0$

$x_7=x_{12}=x_{15}=x_{20}=1$ all other $x_j=0$

Thus the corresponding solutions to the original problem have z=6, and:

$x_1=x_2=x_7=x_{11}=x_{15}=x_{20}=1$ all other $x_j=0$

This says that 6 tests are required to pass through all segments at least once, and the 6 tests should traverse paths 1,2,7,11,15, and 20 as those paths are defined in the matrix U.

As mentioned earlier, the matrix U may sometimes contain some infeasible paths, if the flow relationships in P were not sufficiently obvious for the detection and removal of all infeasible paths. In such a case, one or more infeasible paths may appear in the solution of the binary programming problem. Then, when an attempt is made to construct test data to traverse the paths in the solution, the infeasible paths will be detected. At that time they can be removed from U, and the binary programming problem solved again. The process can be repeated if necessary until a solution free of infeasible paths appears.

The essential features distinguishing the procedure described in this section from the binary programming problem of Section 4.2 are:

In Section 4.2, a matrix F was used to find a maximum incomparable set. F is a square matrix whose dimension is equal to the number of segments in a flowchart. Even in a large flowchart the number of segments might not be more than several hundred, and usually the number of segments is much smaller and manageable in a practical way. F was used to form a maximization problem, in which the solution is the maximum incomparable set of segments, and also the lower bound on the minimum number of tests needed to pass through each segment at least once.

In this section a matrix U is used to find a minimum number of tests. In general, U is not square. The number of rows equals the number of segments in the flowchart, but the number of columns equals the number of feasible paths. Even in a modestly-sized flowchart, the number of feasible paths may be very large and not easy to find. U is used to form a minimization problem, and the solution is the minimum number of feasible paths needed to pass through ("cover") every segment at least once.

Further work is needed to develop methods of deriving U in a manner that is practical for real flowcharts. Also, later an attempt will be made to apply the methods of this section to flowcharts with loops.

## 5.0 Summary and Conclusions

An algorithm for finding the maximum incomparable set has been described. It consists of forming a graph from a flowchart, and then forming an adjacency matrix, performing matrix operations, and finally forming a zero-one integer programming problem and solving it. The size of the maximum incomparable set serves as a lower bound on the number of test cases required, and the members of the maximum incomparable set are the segments through which program test cases must pass. Ideally, it may be possible for the tester to construct test cases so that one and only one will pass through each of the segments of the maximum incomparable set. Sometimes, due to the information contained in the segments, it is not possible to construct such test cases, and the actual number of tests needed to test all segments will be larger than the size of the maximum incomparable set. The size of the maximum incomparable set serves as a lower bound on the number of test cases required. A formula for the upper bound on the minimum number of test cases required was given for loopless flowcharts with two-way decisions. A method for finding the actual minimum number of test cases using binary programming was given.

The relationship between flowchart structure and the number of flowchart paths was examined. It is possible to compute upper and lower bounds on the number of paths in a loopless flowchart from just the number of deciders in the flowchart. Improved bounds can be obtained by decomposing the flowchart and computing bounds for each portion, and then combining the individual bounds in a way that depends on the original decomposition. Further work should be directed toward determination of how the bounds are affected by the method of decomposition, and toward finding the best methods of decomposition.

## REFERENCES


[1] M. Lipow, "Application of Algebraic Methods to Computer Program Analysis," Report TRW-SS-73-10, TRW Software Series, May 1973.

[2] M.L. Shooman, "Analytic Generation of Test Data," unpublished memorandum, December 1974.

[3] G.S. Popkin, "Application of Logical Operations to Finding the Minimum Number of Tests Needed to Verify a Computer Program," Summary of Technical Progress, Software Modeling Studies, January 1, 1976-June 30, 1976, Polytechnic Institute of New York.

[4] R.S. Garfinkel and G.L. Nemhauser, Integer Programming, John Wiley and Sons, New York, 1972, pp. 302,303.

# APPENDIX A

## Unstructured Program and Test Results

```
*PL/C

*OPTIONS IN EFFECT*      TIME=(0,15.00),PAGES=30,LINES=2000,NOATR,NOXREF,FLAGW,NOCMNTS,SORMGIN=(2,72,1),ERRORS=(50,50),
*OPTIONS IN EFFECT*      TABSIZE=5156,SOURCE,OPLIST,NOCMPRS,NOHDRPG,AUXIO=10000,LINECT=60,NOALIST,MONITOR=(UDEF,BNDRY,
*OPTIONS IN EFFECT*      SUBRG,AUTO),NCALL,NONTEXT,DUMP=(S,F,L,E,U,R),DUMPE=(S,F,L,E,U,R),DUMPT=(S,F,L,E,U,R)

T_UN: PROC OPTIONS (MAIN);                                   01  0010   PL/C-R7.6-004 06/16/78 15:35 PAGE   1

 STMT LEVEL NEST BLOCK NLVL  SOURCE TEXT

    1                        T_UN: PROC OPTIONS (MAIN);                                          01  0010
                             /*                                                          */
                             /* INITIALIZE                                               */      01  0020
                             /*                                                          */      01  0040
    2    1         1                 PUT PAGE    LIST('     I   J   K   TYPE OF TRIANGLE');    01  0050
    3    1         1                 PUT SKIP(0) LIST('     _   _   _   ________________');    01  0060
    4    1         1                 PUT SKIP(2);                                              01  0070
                             /*                                                          */      01  0080
                             /*                                                          */      01  0090
                             /*                                                          */      01  0100
    5    1         1           MAIN_LOOP: GET LIST (I,J,K);                                    01  0110
                             /*                                                          */      01  0120
                             /* COUNT MATCHING SIDES                                     */      01  0130
                             /*                                                          */      01  0140
    6    1         1                 MATCH=0;                                                  01  0150
    7    1         1                 IF I=J THEN                                               01  0160
    8    1         1                 MATCH=MATCH+1;                                            01  0170
    9    1         1                 IF I=K THEN                                              010180
   10    1         1                 MATCH=MATCH+2;                                           010190
   11    1         1                 IF J=K THEN                                              010200
   12    1         1                 MATCH=MATCH+3;                                           010210
                             /*                                                          */      02  0010
                             /* SELECT POSSIBLE SCALENE TRIANGLES                        */      02  0020
                             /*                                                          */      02  0030
   13    1         1                 IF MATCH=0 THEN                                          020040
   14    1         1                 GO TO POSSIBLE_SCALENE;                                  020050
                             /*                                                          */     020060
                             /* SELECT POSSIBLE ISOSCELES TRIANGLES                      */     020070
                             /*                                                          */     020080
   15    1         1                 IF MATCH=1 THEN                                          020090
   16    1         1                 GO TO TEST_I_PLUS_J;                                     020100
   17    1         1                 IF MATCH=2 THEN                                          020110
   18    1         1                 GO TO TEST_I_PLUS_K;                                     020120
   19    1         1                 IF MATCH=3 THEN                                          020130
   20    1         1                 GO TO TEST_J_PLUS_K;                                      00140
                             /*                                                          */       0150
                             /* TRIANGLE MUST BE EQUILATERAL                             */       0160
                             /*                                                          */       0170
   21    1         1                 PUT SKIP EDIT (I,J,K,'EQUILATERAL')(X(5),(3)(F(1),X(3)),      0180
                                                                        A(11));                   0190
   22    1         1                 GO TO MAIN_LOOP;                                          0200
                             /*                                                          */       0210
                             /*                                                          */       0220
                             /*                                                          */       0230
   23    1         1           POSSIBLE_SCALENE:                                           03  0010
                                   IF I+J<=K | J+K<=I | I+K<=J THEN                        03  0020
   24    1         1                 GO TO NOT_A_TRIANGLE;                                 03  0030
```

```
T_UN: PROC OPTIONS (MAIN);                                   01  0010   PL/C-R7.6-004 06/16/78 15:35 PAGE   2

 STMT LEVEL NEST BLOCK NLVL  SOURCE TEXT

   25    1    1                    PUT SKIP EDIT (I,J,K,'SCALENE')(X(5),(3)(F(1),X(3)),A(8));  03  0040
   26    1    1                    GO TO MAIN_LOOP;                                            03  0050
                               /*                                          */                   03  0060
                               /*                                          */                   03  0070
   27    1    1               TEST_I_PLUS_J:                                                    03  0080
                                    IF I+J<=K THEN                                            03  0090
   28    1    1                    GO TO NOT_A_TRIANGLE;                                       03  0100
   29    1    1                    ELSE                                                        03  0110
                                    GO TO ISOC;                                               03  0120
   30    1    1               TEST_I_PLUS_K:                                                      30130
                                    IF I+K<=J THEN                                              30140
   31    1    1                    GO TO NOT_A_TRIANGLE;                                       03  0150
   32    1    1                    ELSE                                                            30160
                                    GO TO ISOC;                                                 30170
   33    1    1               TEST_J_PLUS_K:                                                      0180
                                    IF J+K<=I THEN                                              0190
   34    1    1                    GO TO NOT_A_TRIANGLE;                                       03  0200
   35    1    1                    ELSE                                                        03  0210
                                    GO TO ISOC;                                               03  0220
                               /*                                          */                   04  0020
                               /* TRIANGLE IS IMPOSSIBLE                   */                   04  0020
                               /*                                          */                   04  0030
   36    1    1               NOT_A_TRIANGLE:                                                   04  0040
                                    PUT SKIP EDIT (I,J,K,'NOT A TRIANGLE')(X(5),(3)(F(1),X(3)),  04  0050
                                                                          A(14));               04  0060
   37    1    1                    GO TO MAIN_LOOP;                                            04  0070
                               /*                                          */                   04  0080
                               /* TRIANGLE IS ISOSCELES                    */                   04  0090
                               /*                                          */                   04  0100
   38    1    1               ISOC:    PUT SKIP EDIT (I,J,K,'ISOSCELES')(X(5),(3)(F(1),X(3)),A(9));04  0110
   39    1    1                    GO TO MAIN_LOOP;                                            04  0120
   40    1    1               END T_UN;                                                        04  0130

ERRORS/WARNINGS DETECTED DURING CODE GENERATION:

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

```
   I   J   K   TYPE OF TRIANGLE


   4   3   2   SCALENE
   1   1   3   NOT A TRIANGLE
   1   1   2   NOT A TRIANGLE
   2   3   2   ISOSCELES
   2   5   2   NOT A TRIANGLE
   2   4   2   NOT A TRIANGLE
   1   2   2   ISOSCELES
   5   2   2   NOT A TRIANGLE
   4   2   2   NOT A TRIANGLE
   3   3   2   ISOSCELES
   1   1   1   EQUILATERAL
   0   1   2   NOT A TRIANGLE
   1   2   3   NOT A TRIANGLE
   2   0   1   NOT A TRIANGLE
   3   1   2   NOT A TRIANGLE
   1   2   0   NOT A TRIANGLE
   2   3   1   NOT A TRIANGLE
***** ERROR IN STMT    5  END OF FILE REACHED. (EX02)
ABOVE ERROR IS FATAL.  PROGRAM IS STOPPED.
```

# APPENDIX B

## Structured Program and Test Results

```
*PL/C

*OPTIONS IN EFFECT*      TIME=(0,15.00),PAGES=30,LINES=2000,NOATR,NOXREF,FLAGW,NOCMNTS,SORMGIN=(2,72,1),ERRORS=(50,>0),
*OPTIONS IN EFFECT*      TABSIZE=>150,SOURCE,OPLIST,NOCMPRS,NOHDRPG,AUXIO=10000,LINECT=60,NOALIST,MONITOR=(UDEF,BNDRY,
*OPTIONS IN EFFECT*      SUBRG,AUTO),NCALL,NOMTEXT,DUMP=(S,F,L,E,U,R),DUMPE=(S,F,L,E,U,R),DUMPT=(S,F,L,E,U,R)

 TSTRUKT:   PROCEDURE OPTIONS (MAIN);                                   01  0010   PL/C-R7.6-004 06/16/78 15:35 PAGE   1

  STMT LEVEL NEST BLOCK NLVL  SOURCE TEXT

    1                          TSTRUKT:   PROCEDURE OPTIONS (MAIN);                                   01  0010
    2     1          1                   DCL VALID_TRIANGLE RETURNS (CHAR(7));                        01  0020
    3     1          1                   DCL MORE_INPUT CHAR(6) INIT('.TRUE.');                       01  0030
    4     1          1                   DCL MESSAGE (0:3) CHAR(11)                                   01  0040
                                         INIT ('SCALENE','ISOSCELES','ER','EQUILATERAL');             01  0050
                                   /*                                                                 */  01  0060
                                   /* INITIALIZE                                                      */  01  0070
                                   /*                                                                 */  01  0080
    5     1          1                   PUT PAGE    LIST('     I   J   K   TYPE OF TRIANGLE');       01  0090
    6     1          1                   PUT SKIP(0) LIST('     _   _   _   ----------------');       01  0100
    7     1          1                   PUT SKIP(2);                                                 01  0110
                                   /*                                                                 */  01  0120
                                   /* MAIN ROUTINE                                                    */  01  0130
                                   /*                                                                 */  01  0140
    8     1          1              DO WHILE (MORE_INPUT = '.TRUE.');                                 01  0150
    9     1    1     1               GET LIST (I,J,K);                                                01  0160
   10     1    1     1               IF VALID_TRIANGLE = '.TRUE.' THEN                                01  0170
   11     1    1     1               DO;                                                              01  0180
   12     1    2     1                CALL NUMBER_OF_SIDE_MATCHES;                                    02  0010
   13     1    2     1                PUT SKIP EDIT (I,J,K,MESSAGE(MATCHES))(X(5),(3)(F(1),X(3)),A(11));02  0020
   14     1    2     1               END;                                                             02  0030
   15     1    1     1               ELSE                                                             02  0040
                                     PUT SKIP EDIT (I,J,K,'NOT A TRIANGLE')(X(5),(3)(F(1),X(3)),A(14));02  0050
   16     1    1     1              END;                                                              02  0060
                                   /*                                                                 */  02  0066
                                   /* END OF MAIN ROUTINE                                             */  02  0068
                                   /*                                                                 */  02  0070
                                   /* INTERNAL PROCEDURES                                             */  02  0080
                                   /*                                                                 */  02  0090
                                   /*                                                                 */  02  0100
   17     1          1              VALID_TRIANGLE: PROC RETURNS (CHAR(7));                           03  0020
                                   /*                                                                 */  03  0030
   18     2          2                      IF I=J|I=K|I=K|J=K|J=K|J=I THEN                           03  0040
   19     2          2                      RETURN ('.FALSE.');                                       03  0050
   20     2          2                      ELSE                                                      03  0060
                                             RETURN ('.TRUE. ');                                     03  0070
   21     2          2              END VALID_TRIANGLE;                                               03  0080
                                   /*                                                                 */  03  0090
                                   /*                                                                 */  05  0020
   22     1          1              NUMBER_OF_SIDE_MATCHES: PROC;                                      050030
                                   /*                                                                 */  05  0040
   23     2          3                      MATCHES=0;                                                050050
   24     2          3                      IF I=J THEN                                               050060
   25     2          3                      MATCHES=MATCHES+1;                                        050070
   26     2          3                      IF I=K THEN                                               050080
   27     2          3                      MATCHES=MATCHES+1;                                        050090
   28     2          3                      IF J=K THEN                                               050100
```

```
 STMT LEVEL NEST BLOCK MLVL  SOURCE TEXT

   29    2          3                  MATCHES=MATCHES+1;                                  050110
   30    2          3              END NUMBER_OF_SIDE_MATCHES;                          05  0120
                              /*                                          */              050130
   31    1          1          END TSTRUKT;                                             050140

ERRORS/WARNINGS DETECTED DURING CODE GENERATION:

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

```
 I   J   K   TYPE OF TRIANGLE

 1   1   3   NOT A TRIANGLE
 1   1   2   NOT A TRIANGLE
 1   3   1   NOT A TRIANGLE
 1   2   1   NOT A TRIANGLE
 3   1   1   NOT A TRIANGLE
 2   1   1   NOT A TRIANGLE
 2   3   4   SCALENE
 4   3   2   SCALENE
 1   1   1   EQUILATERAL
 2   2   3   ISOSCELES
***** ERROR IN STMT    9  END OF FILE REACHED. (EX02)
ABOVE ERROR IS FATAL.  PROGRAM IS STOPPED.
```

# APPENDIX C

## Simplest Structured Program and Test Results

```
*PL/C

*OPTIONS IN EFFECT*      TIME=(0,15.00),PAGES=30,LINES=2000,NOATR,NOXREF,FLAGW,NOCMNTS,SORMGIN=(2,72,1),ERRORS=(50,50),
*OPTIONS IN EFFECT*      TABSIZE=5156,SOURCE,OPLIST,NOCMPRS,NONORPG,AUXIO=10000,LINECT=60,NOALIST,NONITOK=(QUOTE,BNDRY,
*OPTIONS IN EFFECT*      SUBRG,AUTO),NCALL,NONTEXT,DUMP=(S,F,L,E,U,R),DUMPE=(S,F,L,E,U,R),DUMPT=(S,F,L,E,U,R)

TSINP:     PROC OPTIONS (MAIN);                                   01  0010   PL/C-R7.6-004 06/16/78 15:35 PAGE   1

 STMT LEVEL NEST BLOCK NLVL  SOURCE TEXT

    1                        TSINP:     PROC OPTIONS (MAIN);                                          01  0010
    2    1         1                    DCL MORE_INPUT CHAR (6) INIT ('.TRUE.');                      01  0020
                             /*                                                        */              01  0060
                             /* INITIALIZE                                             */              01  0070
                             /*                                                        */              01  0080
    3    1         1                    PUT PAGE    LIST('     I   J   K   TYPE OF TRIANGLE');      01  0090
    4    1         1                    PUT SKIP(0) LIST('     _   _   _   ----------------');      01  0100
    5    1         1                    PUT SKIP(2);                                                01  0110
                             /*                                                        */              01  0120
                             /* MAIN ROUTINE                                           */              01  0130
                             /*                                                        */              01  0140
    6    1         1                    DO WHILE (MORE_INPUT = '.TRUE.');                           01  0150
    7    1    1    1                     GET LIST (I,J,K);                                           01  0160
    8    1    1    1                     L=1; M=J; N=K;                                              01  0165
   11    1    1    1                     IF I>J THEN                                                 01  0170
   12    1    1    1                     CALL INVERT (I,J);                                          01  0180
   13    1    1    1                     IF J>K THEN                                                 01  0190
   14    1    1    1                     CALL INVERT (J,K);                                          01  0200
   15    1    1    1                     IF I>J THEN                                                 01  0210
   16    1    1    1                     CALL INVERT (I,J);                                          01  0220
   17    1    1    1                     IF I+J<=K THEN                                              01  0230
   18    1    1    1                     PUT SKIP EDIT (L,M,N,'NOT A TRIANGLE')(X(5),(3)(F(1),X(3)),A(14)); 01  0240
   19    1    1    1                     ELSE                                                        01  0250
                                           IF I=J THEN                                               02  0010
   20    1    1    1                       IF J=K THEN                                             01  0020
   21    1    1    1                       PUT SKIP EDIT (L,M,N,'EQUILATERAL')(X(5),(3)(F(1),X(3)),A(11));
   22    1    1    1                       ELSE                                                    02  0040
                                            CALL OUTPUT_ISOSCELES;                                 02  0050
   23    1    1    1                     ELSE                                                      02  0060
                                            IF J=K THEN                                            02  0070
   24    1    1    1                       CALL OUTPUT_ISOSCELES;                                  02  0080
   25    1    1    1                       ELSE                                                    02  0090
                                            PUT SKIP EDIT (L,M,N,'SCALENE')(X(5),(3)(F(1),X(3)),A(7)); 02  0100
   26    1    1    1                   END;                                                        02  0060
                             /*                                                        */              02  0066
                             /* END OF MAIN ROUTINE                                    */              02  0068
                             /*                                                        */              02  0070
                             /* INTERNAL PROCEDURES                                    */              02  0080
                             /*                                                        */              02  0090
                             /*                                                        */              02  0100
   27    1         1                    INVERT: PROC (INA,INB);                                     02  0120
   28    2         2                            ITEMP=INA;                                          02  0130
   29    2         2                            INA=INB;                                            02  0140
   30    2         2                            INB=ITEMP;                                          02  0150
   31    2         2                    END INVERT;                                                 02  0160
                             /*                                                        */              02  0170
                             /*                                                        */              02  0180
```

```
TSIMP:     PROC OPTIONS (MAIN);                                   01  0010   PL/C-R7.6-004 06/16/78 15:35 PAGE   2

 STMT LEVEL NEST BLOCK NLVL  SOURCE TEXT

   32    1          1            OUTPUT_ISOSCELES: PROC;                                                  02  0190
   33    2          3                 PUT SKIP EDIT (L,N,N,'ISOSCELES')(X(5),(3)(F(1),X(3)),A(9));
   34    2          3            END OUTPUT_ISOSCELES;                                                    02  0210
   35    1          1          END TSIMP;                                                                 02  0220

ERRORS/WARNINGS DETECTED DURING CODE GENERATION:

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

```
   I   J   K   TYPE OF TRIANGLE

   1   1   1   EQUILATERAL
   2   3   2   ISOSCELES
   3   2   3   ISOSCELES
   2   3   4   SCALENE
   9   5   4   NOT A TRIANGLE
   8   4   3   NOT A TRIANGLE
***** ERROR IN STMT    7  END OF FILE REACHED. (EX02)
ABOVE ERROR IS FATAL.  PROGRAM IS STOPPED.
```