# DFSCRIP: An Implementation of a Single-pass Algorithm for the Critical Path Method

By

Subramanian Srinivasan

Master of Engineering

Birla Institute of Technology & Science

Pilani INDIA

1973

Submitted to the Graduate Faculty of the

Department of Management

College of Business Administration

Oklahoma State University

in partial fulfillment of

the requirements for the degree of

MASTER OF BUSINESS ADMINISTRATION

JAN 1984

Name: Subramanian Srinivasan Date of Degree - May 1984

Institution: Oklahoma State University

Location: Stillwater

Title of Study: DFSCRIP: An implementation of a Single-pass Algorithm for the Critical Path Method

Pages in study: 60 Candidate for the Degree of Master of Business Administration

Major Field: Business Administration


Scope and Method of Study:
The scope of this report is the development of comparable computer programs for a new single-pass algorithm and the conventional method for finding the critical path in a project network and to study their relative efficiencies.Adequate examples are used for being able to draw conculsions about the complexities of the algoritms.Since the nature of the study is theoretical,the examples used are arbitrary networks and not drawn from real life situations.

Conclusions:
The results of the computer executions indicated that the complexity of the new algorithm is linear,requiring time proportional to the number of activities in the network.However, the algorithm turned out to be slower than expected and was only marginally faster than the conventional algorithm.The storage requirements of the two algorithms were also proportional to the number of activities,but the new algorithm needed somewhat more storage.These results led to the conclusion that both the algorithms are of approximately equal overall efficiencies.

DFSCRIP: An Implementation of a Single-pass Algorithm for the Critical Path Method

Report Approved:

Advisor

Director of Graduate Studies

Head, Department of Management

## ACKNOWLEDGEMENT

I would like to express my gratitude to Prof.Locks not only for supervising this work but also for all the encouragement he has given me during the course of my association with the project.

I would also like to thank the Office of Business Economics and Research for providing excellent facilities, and particularly Kathy Bolstead for keeping the terminals warm and offering a ready helping hand. Thanks also to my colleagues on the project who have all contributed to this paper in some way or the other.

Finally, I would like to express my appreciation for my wife, Maithili, who has been a tremendous support and source of encouragement all along.

TABLE OF CONTENTS

# DFSCRIP: AN IMPLEMENTATION OF A SINGLE-PASS ALGORITHM FOR THE CRITICAL PATH METHOD

## 0.0 ABSTRACT

This paper is concerned with the development and documentation of a computer program for a new algorithm developed by M.O.Locks for finding the critical path. The algorithm is a depth-first search that requires only one pass through the network,while the conventionally used method requires two passes.The time complexity of the algorithm is linear in the number of activities in the network.The two methods are compared with the help of example problems containing 25 to 300 activities.

An important feature of the computer program is modularization that enables linking up smaller networks to solve large problems with relatively less computer storage requirements.Another feature is the construction by the program of a linked list for storing information about the network.The program is written in the PL/1 language.

Also introduced in this paper is Event Float,a measure that is different from the other commonly known floats.

--------------------

This project was supported in part by the Air Force under project number AFOSR-82-0251

# 1.0 INTRODUCTION

## 1.1 Preview of project-network analysis methods

The key to successful project execution is proper planning,scheduling and control. Project Evaluation and Review Technique(PERT) and Critical Path Method (CPM) are two methods that enable these three phases to be carried out effectively.Until the mid-fifties the bar chart,also known as the Gantt chart,was the primamry tool available for management.In 1958 PERT

CPM identifies the sequence of activities through the network along which no slack can be permitted if the project is to be executed in the specified time.The crititcal path(CP) is the longest path through the network.Though CPM and PERT are very similar,PERT relates to probabilistic networks and originally did not concern very much with the cost element.CPM,on the other hand,requires that time elements be known with certainty and is extensively used in resource allocation problems.A simple method for cost-time trade off that is suitable for hand calculation is described by Siemens(3) and a computational procedure based on network flow theory has been developed by Fulkerson(4).

## 1.2 Backtracking and Depth-first search

Backtracking is a technique that has been derived and used independently over the years in different contexts in combinatorial theory.A very well known application in Operations Research is the Branch and Bound algorithm in integer programming. A generalized treatment of the properties of backtracking has been provided by Golomb and Baumert(5).The search in backtracking proceeds in a predetermined manner and everytime an end condition is reached, the search backs up to a previously investigated state and resumes from there.

Depth-first search(DFS) is a graph theroy technique that was introduced by Hopcroft and Tarjan and subsequently elaborated upon by Tarjan(6). The rule governing the search is such that the search always proceeds towards the terminal.Backing up occurs when the terminal is reached and thus backtracking is always a part of DFS.Tarjan's treatment of DFS is general with respect to the type of graph it deals with.

1.3 Berztiss' work in DFS

In connection with directed graphs,of which a project activity network is an example,Berztiss has broken up the DFS tree into atomic units(7). Each atomic tree represents an event and all its successors. Berztiss has used this atomic approach to store information about the network in 'arc' and 'node' tables.In the node table each event is viewed as the starting event of an activity and a set of data is stored from that perspective.In the arc table information pertaining to each activity is stored. The atomic trees are integrated into a larger K-tree representing the network. Correspondingly,the initial arc and node table are reconstructed to obtain the data storage structure in terms of a final node and arc table.

Some of the advantages claimed by Berztiss are the ability to use standard tree traversal algorithms and the ability to write the algorithm clearly in a non-recursive manner.In this report the more direct approach used by Tarjan is followed and the data storage structure appears to be much simpler and to require lesser data entry effort.

### 1.4 Applications of DFS

DFS has innumerable applications in graph theory.Tarjan and Hopcroft have used DFS to determine planarity and isomorphism of graphs. DFS has been applied to find system reliability by Satyanaryana(8). This paper has resulted from implementing on the computer an algorithm developed by Locks for establishing the critical path(9). A comparison of the merits of 20 commercially available software programs for executing CPM has been carried out by Mahler and Smith(10).

### 1.5 A New Float Measure

Associated with the critical path method are the calculation of various float measures.There are four generally known floats - Total Float,Free Float,Independent Float and Safety Float(11).All these float measures are activity oriented and they specify the slack that can be permitted in an activity under various limiting conditions.An event oriented float,introduced by Locks(9),is described in this paper.

## 2.0 CPM:THE CONVENTIONAL METHOD AND DEPTH-FIRST SEARCH

In this section the conventional method is reviewed briefly and then DFS,as applied to a project network,is described in detail.

### 2.1 The conventional method

The conventional method requires one pass in each direction through the network to find the critical path.In the forward pass the earliest start time(ES) for all the activities are calculated and in the backward pass the latest finish time(LF).The earliest occurrence time(EET) for an event is the ES of all activities succeeding it and the latest occurrence time(LET) is the LF of all the preceding activities.The critical path is the sequence of activities connecting a critical sequence of events which all have their EET and LET equal.There are innumerable texts describing the method but a recent one with a number of references is by Phillips and Garcia-Diaz(12).

### 2.2 DFS applied to CPM

The application of DFS to determine the critical path in a project network ,as develped by Locks,is described here.

#### 2.2.1 The rooted search tree

In the process of conducting the DFS a rooted tree equivalent of the original network is constructed.The edges in the tree and the activites in the network have a one to one correspondence.That is,each edge represents one and only one activity.The nodes,on the other hand,are partial events in the sense that each node represents the completion of only one activity preceding it.The tree is constructed by adding an edge and a node at the end of that edge for each activity explored.If a set of activities start from one event then the corresponding set of edges also start from the same node.

2.2.2 The search rule

The search starts from the root of the tree which corresponds to the starting event on the network.Exploration is governed by the rule that at every stage in the search,the next event to resume search from is the most recently reached event which still has unexplored activities starting from it.The most recent event is necessarily the furthest down the network in the direction of the pass.Thus the search proceeds quickly downwards,reaching the terminal event.Then backing up to the last event reached, exploration continues downwards till the terminal or an already visited event is encountered.The search tends to proceed along the network rather than across it.

2.2.3 An example by backward search

DFS can be conducted in either direction,from the starting event or from the ending event.The network in Figure 1 is used as an example to describe DFS.The backward search is employed rather than the forward search for certain explanational convenience. The DFS search tree for this network is shown in Figure 2.Node 1 on the tree represents event 7.Activities 5-7 and 6-7 terminate at 7 and correspondingly nodes 2 and 3 are added on the tree.The last event reached was 6.Further search from there leads to events 3 and 4.Nodes 4 and 5 represent these events.Proceeding from 4 event 1 is reached.Node 6 is added to represent event 1.The last event reached with unexplored activities is 3.Events 1 and 4 are reached from 3 and are represented by nodes 7 and 8.

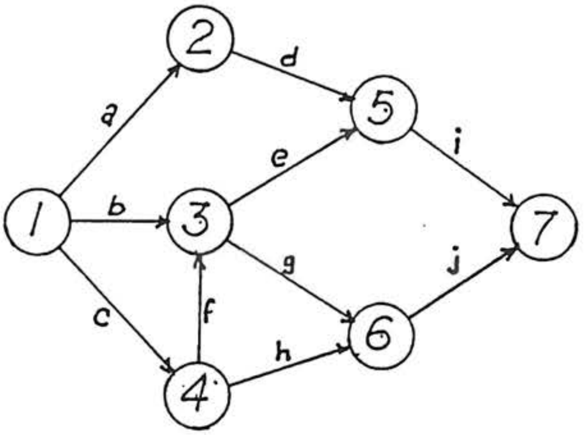



Figure 1 -Example Network

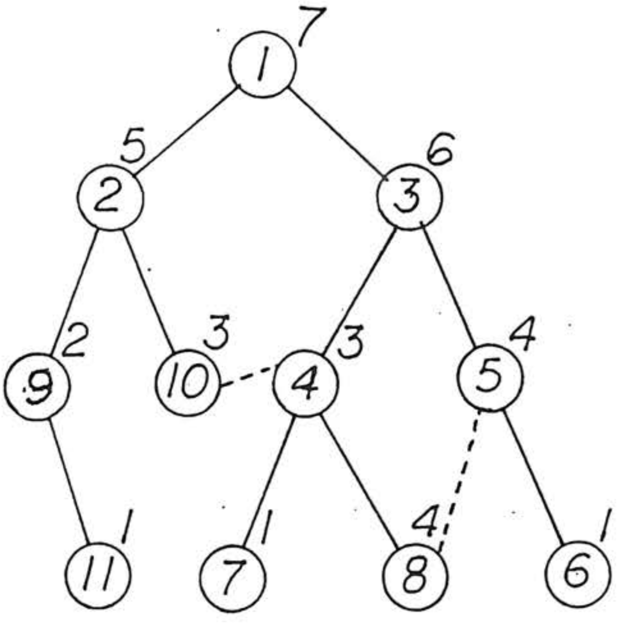



Figure 2 - DFS Tree for Network in Figure 1

Further search continues from event 5 resulting in nodes 9,10 and 11.Since each activity is explored exactly once, the tree contains n edges and n+1 nodes,where n is the number of activities.

The critical path is now obtained by traversing the tree.The EET for event 4 is {EET(1)+ duration of 1-4},which is just c,EET(1) being 0. EET(3) is given by max{EET(1)+ duration 1-3, EET(4)+ duration 4-3}, which is max{b, c+f}.Going up the tree,EET(6)= max{EET(3)+ duration 3-6,EET(4) +duration 4-6}.Thus EET(6)= max{g+ max{b,c+f} ,c+h}.Likewise traversing up the left side of the tree we obtain EET(5) as max{EET(3)+ duration 3-5, EET(2)+duration 2-5}.EET(2) is a.Finally, EET(7)=max{EET(5)+ duration 5-7, EET(6)+ duration 6-7}.Proceeding from the leaves to the root all these expressions can be evaluated and EET(7) gives the project time.

### 2.2.4 Retracing

Each sequence of edges on the tree is a partial path.The critical path is the longest path from the root to any leaf corresponding to the starting event 1.The process of comparison at each node to find the EET of the corresponding event also enables to identify the edge to be followed for the longest partial path.For example at node 1 if EET(5)+duration 5-7 is greater than EET(6)+duration 6-7 then the longest path is along edge 7-5.With this information available for every event ,it is easy to trace the critical path starting from the root.If a node is a leaf,then a jump is to be made.Due to the backtracking procedure corresponding to each event there can be only one node which is not a leaf,the one corresponding to the first visit to that event. A jump is then to be made to this node and the tracing continued.In Figure 2 node 5 corresponds to the first visit to event 4.Hence if this happens to be on the critical path,a jump would have to be made node 5 whenever a leaf corresponding to event 4 is encountered.If 1-4-3-5-7 were the critical path,then the path would be traced along nodes 1-2-(10-4)-(8-5)-6.

### 2.2.5 Direction of Pass and Event times

An interesting consequence of the nature of DFS is that during the forward pass we obtain the LETs and during the backward pass the EETs. This is because once a path from any event to the terminal is established, that path is never traversed again.The length of that path is fixed right at the first traversal.On the other hand,any number of paths to that event from the source can be established until the search is completed.Thus,in the forward pass the longest paths from intermediate events to the ending event is obtained and in the backward pass,the longest path from the starting event to intermediate events.

## 3.0 SALIENT FEATURES OF THE COMPUTER PROGRAMS

Two important features have been incorporated in the computer programs.

### 3.1 Modularization

The most important user friendly feature of the computer programs is their ability to integrate modular networks into a supernetwork.Large, complex projects may be made up of smaller independent projects.Solving the entire project as a single problem would require proportionately high computer storage requirements.With the modularization feature,the storage required is dependent only upon the size of the largest module. Problems with 25,50,100 and 300 activities, presented in section8.0, have been integrated and the complete problem,with 475 activities,required only 76 kbytes,the same memory required by the 300 activity problem.However, in order to be able to use the feature it must be possible to break up larger network into smaller modules,each with one starting and one ending event.This may not be possible with all networks.

The concept of modules has been known in a different form as a 'Hammock'. A Hammock, or a Summary,activity is a single activity that represents a section of a network. PREMIS(13) calculates the duration of the Hammock activity when the extreme events of a section are specified. The programs in this report performs the reverse function,they integrate the Hammock activities into bigger networks.

### 3.2 Representation of network data

The first step in implementing the computer program is to arrive at a suitable method of representing the network.The simplest way to do this be to use an NxN matrix,N being the number of events.Entry in row k and coloumn j represents the duration of activity k-j.However,the resulting matrix would be very sparse,resulting in unnecessary use of memory and time consuming search.A simple, linked list form of data storage has been used in this paper.Ashbrook and Zinn(14) have described another form of linked list but their method requires identifying the activities by numbers.The additional work may not be warranted for the type of network this paper is concerned with.

3.2.1 Linked list

For each event in the network information about all immediately succeeding activities are stored in a data structure constructed by the program and referred to here as a 'card'.The cards are identified by a serial number and contain three fields.One field contains the succeeding event, another contains the duration of the corresponding activity,and the third field is a pointer that contains the serial number of the next card on the list.For each event there is a 'header' which contains a pointer. All cards pertaining to one predecessor event are linked linearly and the whole chain is linked to the header.The last card in the chain is identified by a 0 in its pointer field.Figure 3 shows a small network and the linked list for it.

As the input data is read cards are drawn(in serial order) and written in and linked up.To identify the successor activities of,say,event 3, the pointer field of header 3 is read. The 4 in that field implies that the first card on the linked list is

4.The 5 in the pointer field of 4 leads to card 5.The 0 in card 5 implies that there are no other activities starting from event 3.

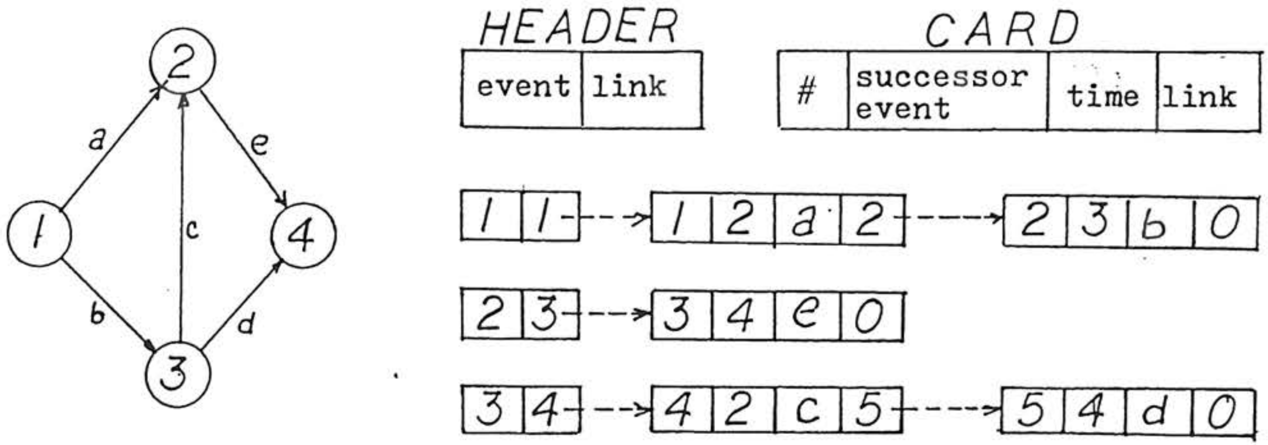



Figure 3 - Example for Linked List Construction

3.2.2 Bi-directional list

The list explained above is adequate if only the forward pass is to be made.For a reverse pass lists for all preceding activities for each event is required.The header then contains two pointers,the backward pointer identifies the first card on the predecessor list.Both lists are are constructed simultaneously as the input data is read.

3.2.3 Savings in memory

When both lists are constructed there will be 2M cards,M being the number of activities.For N events there will be N headers.With 4 elements in each card and 3 elements in each header,a total of 4M+3N elements are required.If M=2N then less than 20N elements are required.Except in very small problems this is very much smaller than $N^2$.The other major advantage is that the search for information is limited to small lists and avoids unnecessary scanning.

## 4.0 COMPUTER PROGRAM OUTLINE

This section briefly describes the DFSCRIP(Depth First Search for CRItical Path) computer program.The CONCRIP(CONventional method for CRItical Path) program has been written on a very similar basis, requiring an identical input data, but is not described here as it is already well known.

The programs have been written in PL/1 language.The major advantages of PL/1 is the feasibility of using structured variables in which a number of variables can be grouped together under a common name.This facility is extremely useful in creating the linked list and the tree structure.The second advantage,though minor,is the ability to identify modules by names rather than numbers.

### 4.1 DFSCRIP

The main procedure is named DFSCRIP and has 6 internal procedures.The main procedure reads the names of the modules and the number of events in each module and invokes the procedure 'SEARCH'.SEARCH has three functions.First it reads the input data and constructs the linked list. Then it calls two internal procedures 'BUILD_TREE' and 'CRITICAL_PATH' to execute DFS.Another procedure,'CP_ROUTE',is invoked to trace the critical path.The third section calculates the various float values.

BUILD_TREE constructs the tree structure without calculating the event times,the tree configuration being independent of the activity times. There are two arguments to this procedure,a node and the event it represents.The procedure goes through the linked list and for each successor event it creates a node at the next level,a child, and records the relationship between the nodes.The procedure is recursive so that it proceeds down the network by itself.Thus,if the procedure is called by specifying the starting event and the root,the entire DFS tree is constructed.

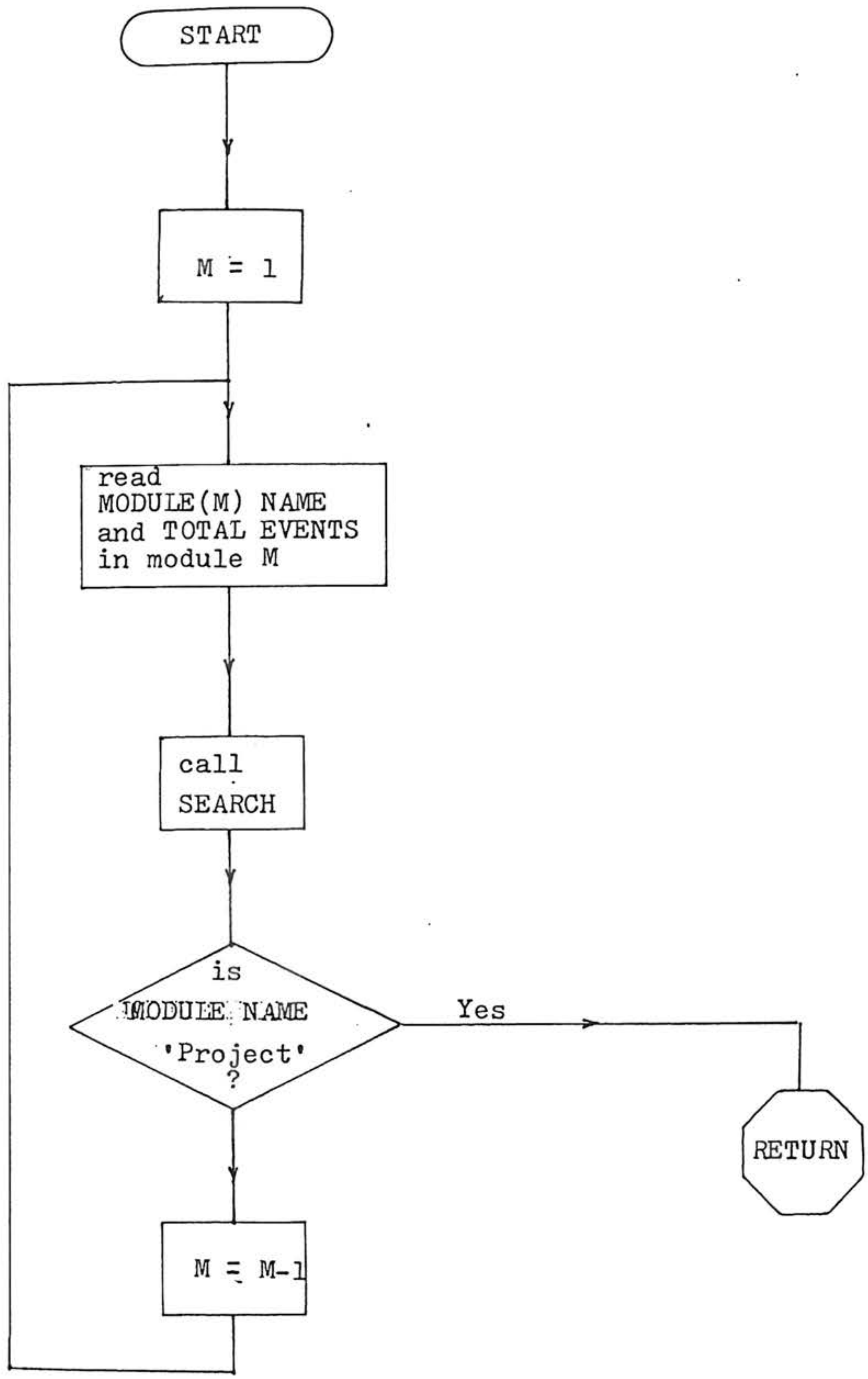
CRITICAL_PATH then calculates the event times by traversing the tree. For a node specified as its argument it identifies the successor events and their event times and then makes the appropriate calculations.Since calculations are to be carried out upwards through the tree,the procedure has been made recursive.For each event the successor event that results in the greatest partial path is recorded.The event time is output as an argument.A third argument specifies the direction of the search.

CP_ROUTE traces the longest path(s) from the event specified as its argument to the terminal. If the event specified is the starting event,then the critical path is obtained.

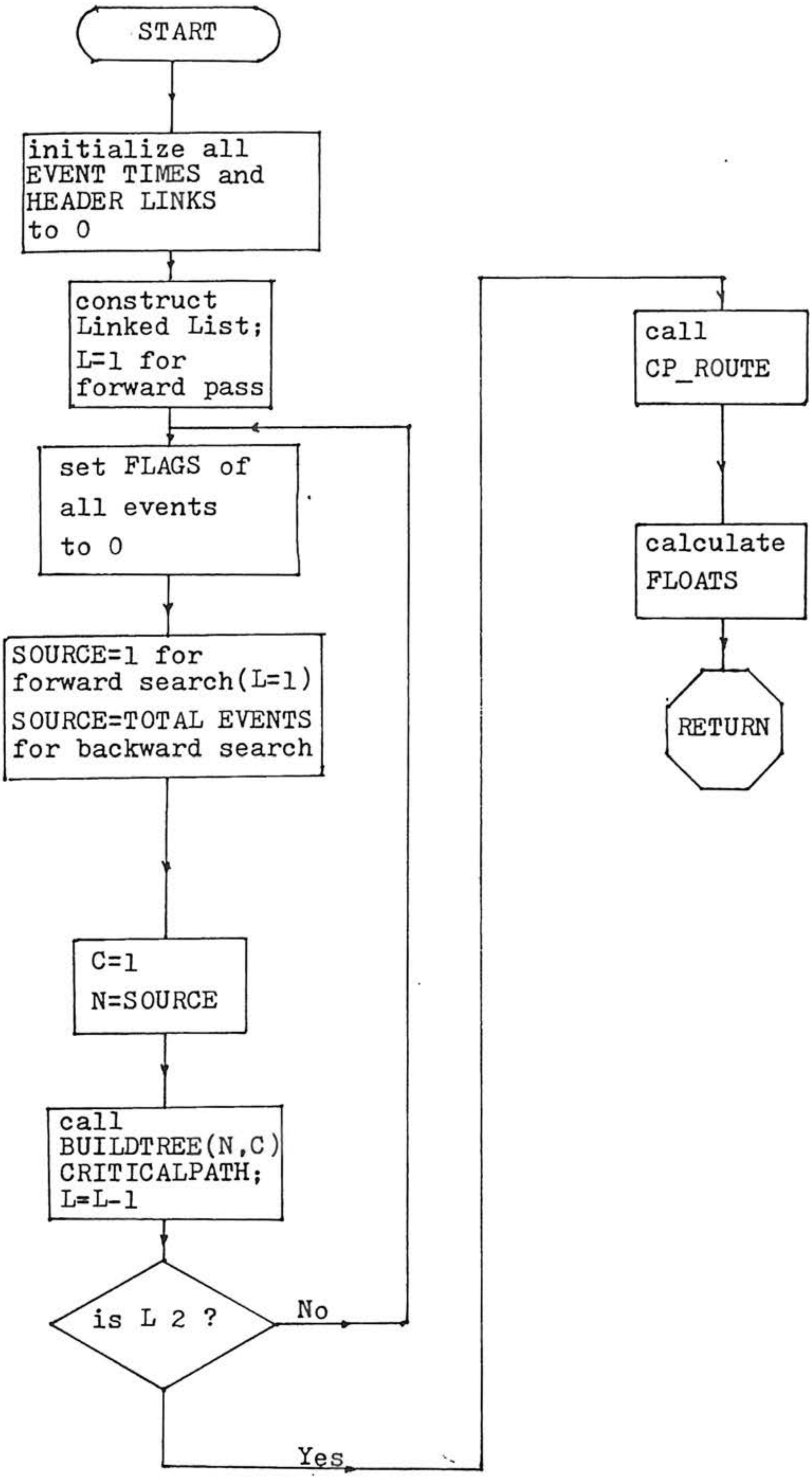
Two other procedures are used in constructing the linked list.TRACE identifies the last card in the linked list for the event argument specified.IDENTIFY provides a numerical identification for modules referred to by a name.Though the user specifies an alpha-numeric name for the modules,the computer assigns a number for each module for internal identification.When a module name is encountered,a proper connection has to be made.

Flow charts for the DFSCRIP ,SEARCH ,BUILD_TREE and CRITICALPATH procedures are presented in the following pages.Flow charts for the other three are omitted because they are very simple procedures,just tracing small sequences of numbers. Complete listings of the programs for the two algorithms are provided in the appendix.

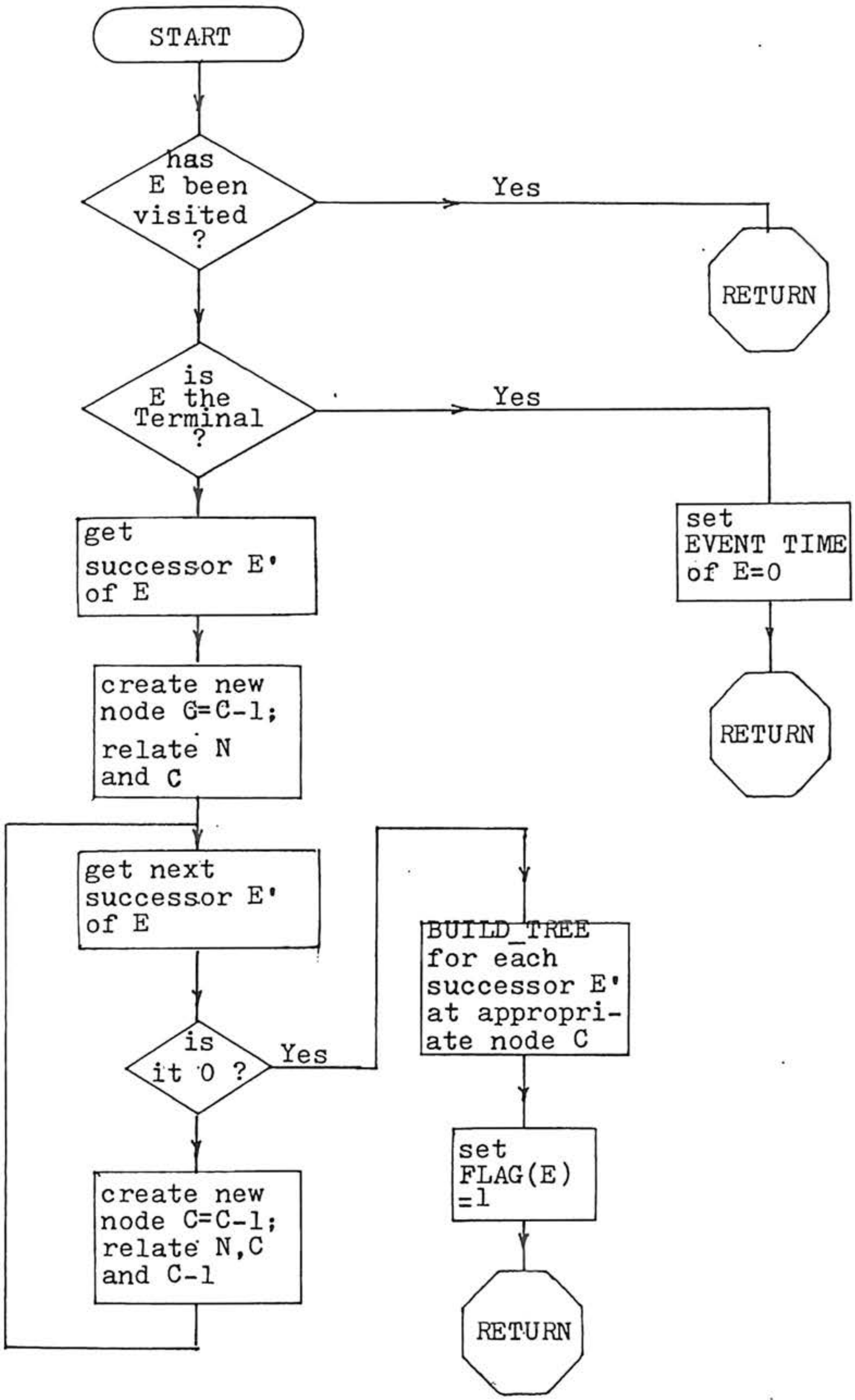
Flow chart for DFSCRIP



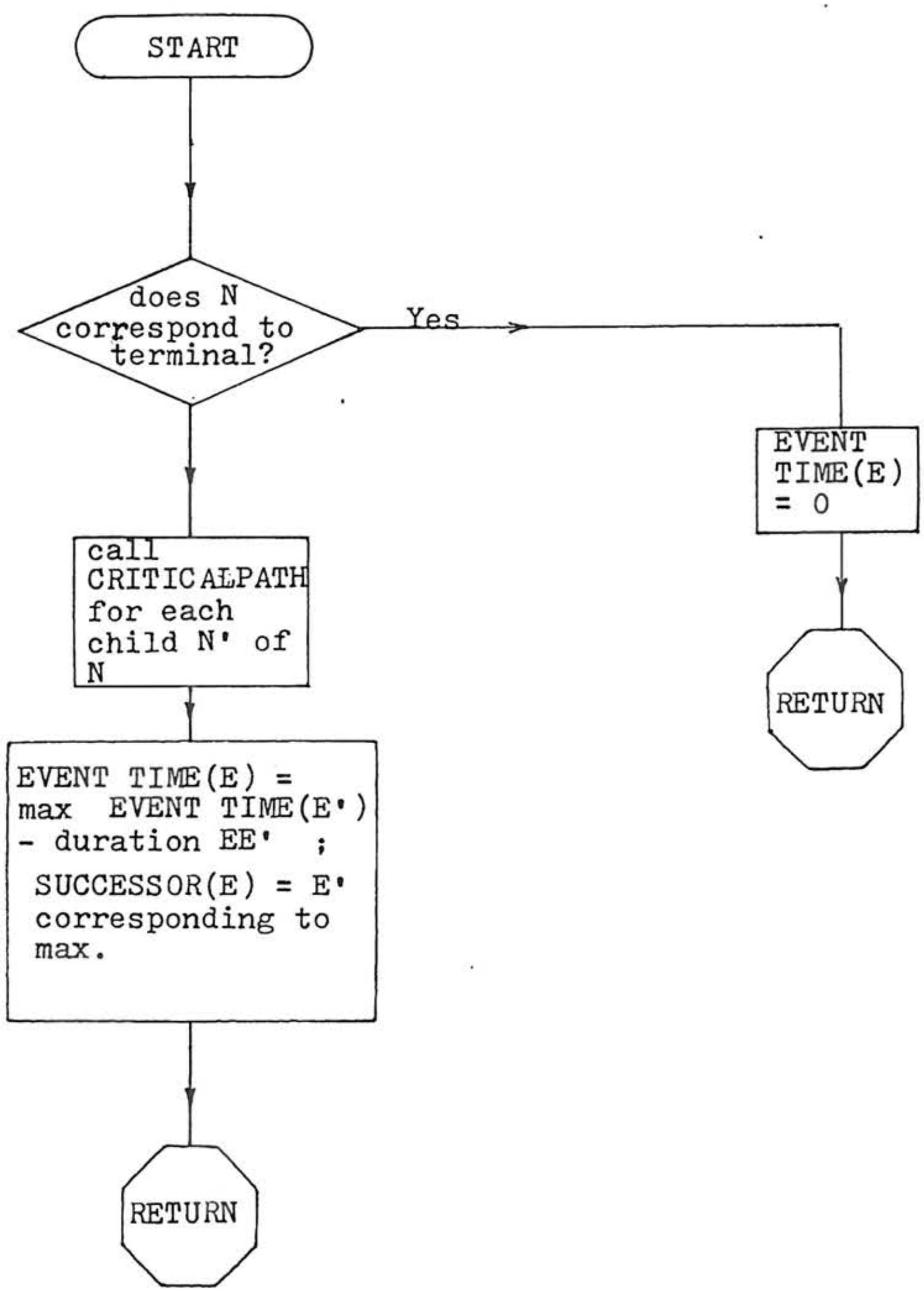
Flow chart for SEARCH



Flow chart for BUILD_TREE(E,N) RECURSIVE



Flow chart for CRITICALPATH(N,EVENT TIME,LINK) RECURSIVE



# 5.0 INSTRUCTIONS FOR USING THE COMPUTER PROGRAMS

Both the computer programs DFSCRIP and CONCRIP perform the same functions.Input and output data format are absolutely identical except in one small detail in the output printout.Both programs can integrate modules into supernetworks.

## 5.1 Inputting Data

All data can be entered in free format.That is,the entries need not be made in specific fields.Only, they must be separated by a comma or atleast one blank.The programs provide for identifying modules by any alpha-numeric character.

The program requires the data pertaining to each module to be entered first, followed by the data for the supernetwork.Examples of input data for some largle modules and supernetwork are presented in section 8.1

### 5.1.1 Module Data

The data concerning a module is divided into two sections.First,there is a header card containing the name of the module and the number of events in the module.The name can contain a maximum of 15 characters, alphabets,numbers or other symbols including blanks.The name is entered first within single quotes(').The second section consists of 'event activity' cards.For each event there is one card with the event number followed by each successor event and the corresponding activity duration. If necessary continuation cards may be used,without any special characters to identify them.The end of the entries of successor events and times is indicated by entering a 0. There will be successor event list for each event except the finishing event.Dummy activities are entered like other activities with a time of 0.

5.1.1.1 An Example

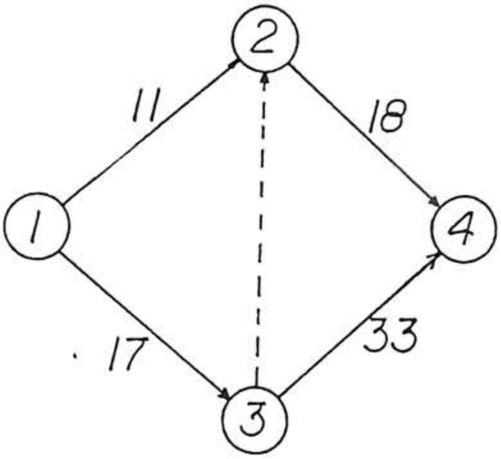



Figure 4 - Example for Module Data Input

For the network in Figure 4 data is to be entered as follows;

'simple assembly' , 4

1 2 11 3 17 0

2 , 4 . 18 0

3 4 33 2 0 0

The job has been called Simple Assembly and has 4 events.From the second card we infer that event 1 is succeeded by event 2 with duration of activity 1-2 equal to 11,and by event 3 with duration of activity 1-3 equal to 17.It may be noted that 3-2 is a dummy activity.

5.1.2 Supernetwork Data

Data entry is identical to that for modules.However,the name MUST be entered as PROJECT.Further,since the critical path of the modules is not known,the activity times must be replaced by the module names.

5.1.2.1 Example for Module Integration

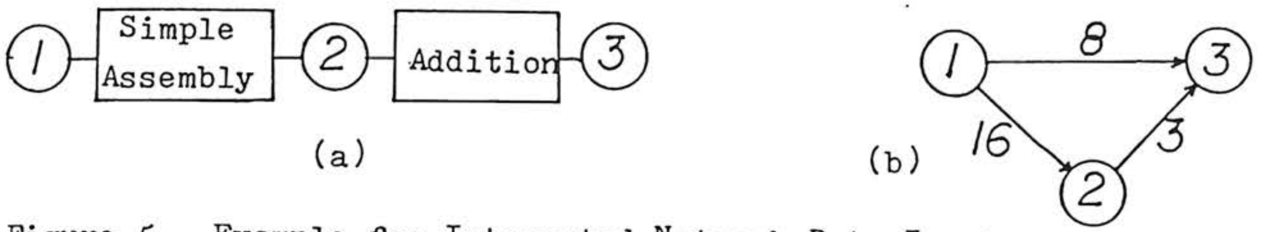



Figure 5 - Example for Integrated Network Data Input

The supernetwork in Figure 5a has two modules, Simple Assembly in Figure 4 and another module, Addition, shown in Figure 5b. The data input in this case would start with the 4 lines of data given in section 5.1.1.1 for Simple Assembly followed by the data for the other module and integrated network as given below.

```
 'Addition'  3
1  2  16  3  8  0
2  3  3  0
'Project'  3
1  2  'Simple Assembly' 0
2  3  'Addition'  0
```

In each module and in the supernetwork the events must be numbered consecutively from 1 without any missing numbers.

5.1.3 Single Module Networks

If the problem has only one module, then data entry is done as in section 5.1.1 except that the name must be entered as PROJECT.

5.2 Output Information

The output information is quite obviously understood. However, in the two programs there is a small difference in the manner in which the critical path is printed. CONCRIP prints out all critical events in numerical order. DFSCRIP prints out the correct sequence of events along the critical path. In the case of multiple paths,DFSCRIP starts again from the event at which multiple paths are encountered.For example,in Module 1 in section 8.0 there are two critical paths 1-4-3-2-5-9-12-13-15 and 1-4-8-7-11-10- 13-15.Multiplicity occurs at event 4 and the second sequence is printed starting from 4 as 1-4-3-2-5-9-12-13-15-4-8-7-11-10-13-15.

5.3 Limits on the Size of Problems

The 300 activity problem presented in section 8.0 required about 76 kbytes of computer memory.It has not been possible to calculate or observe the core requirements for large problems. However, the computer statistics provided in section 6.3 show that on a mainframe computer with several million bytes of memory the size of the problem that can be attempted would be practically unlimited.

In the case of modular networks the size of the problem that can be handled is dependent upon the size of the largest module.This program has been written to handle upto 50 modules.

## 6.0 EVALUATION OF ALGORITHMS

The basis for comparing two algorithms are the execution times and the computer core space required by them. The programs were run on an IBM 3081D system.

### 6.1 Time complexity of DFSCRIP

Time is required for building the linked list and for executing the search.DFSCRIP requires the construction of only the forward linked list.Consider a problem with M activities and N events.Further,let a and b be the times taken by the computer for making an addition and a comparison operation respectively.For the data storage M cards are to be linked up and written in requiring a time of kM,where k is a constant of proportionality. Each activity is explored once,requiring one addition operation and one comparison operation to find the correct ES.The total time taken for these two operations is M(a+c). a and c can be considered equal and the total time taken is then (k+2a)M, which is of linear order in M.

### 6.2 Time Complexity of CONCRIP

The conventional algorithm requires the construction of a backward list also and the time taken for data storage is 2kM. Each activity is explored once,but two passes are required by the algorithm. Further,a subtraction is required for finding the slacks at the events. The total time consumed by the algorithm is then 4aM+2kM+aN. Generally N is somewhat smaller than M and, therefore, the time taken is slightly less than (2k+5a)M.

### 6.3 DFSCRIP vs CONCRIP

Both algorithms are linear complexity and DFSCRIP appears to be theoretically superior. However,the actual time taken by a program is also dependent upon the number of 'bookkeeping' operations.In the case of CONCRIP these operations are merely testing of flags to check if an event has been visited or not. A lot of more time is required by DFSCRIP in the process of building the tree structure.The actual execution times taken by the programs have been plotted as functions of the number of activities in Figure 6. Expectations of linear relationships are confirmed by the plots and DFSCRIP is found to be marginally faster.

Figure 7 shows the core requirements for the two programs as functions of the number of actitivities.These are also close to being linear.But the conventional method requires very few variables to be stored in the process of the search.The tree structure in DFS resulted in about 65% greater core requirement.This difference in the two algorithms can be reduced to about 25% by writing the algorithm non-recursively, but that would result in the execution time increasing by 10% over that of the recursive procedure.

In conclusion, both algorithms are observed to be of equal overall efficiency.

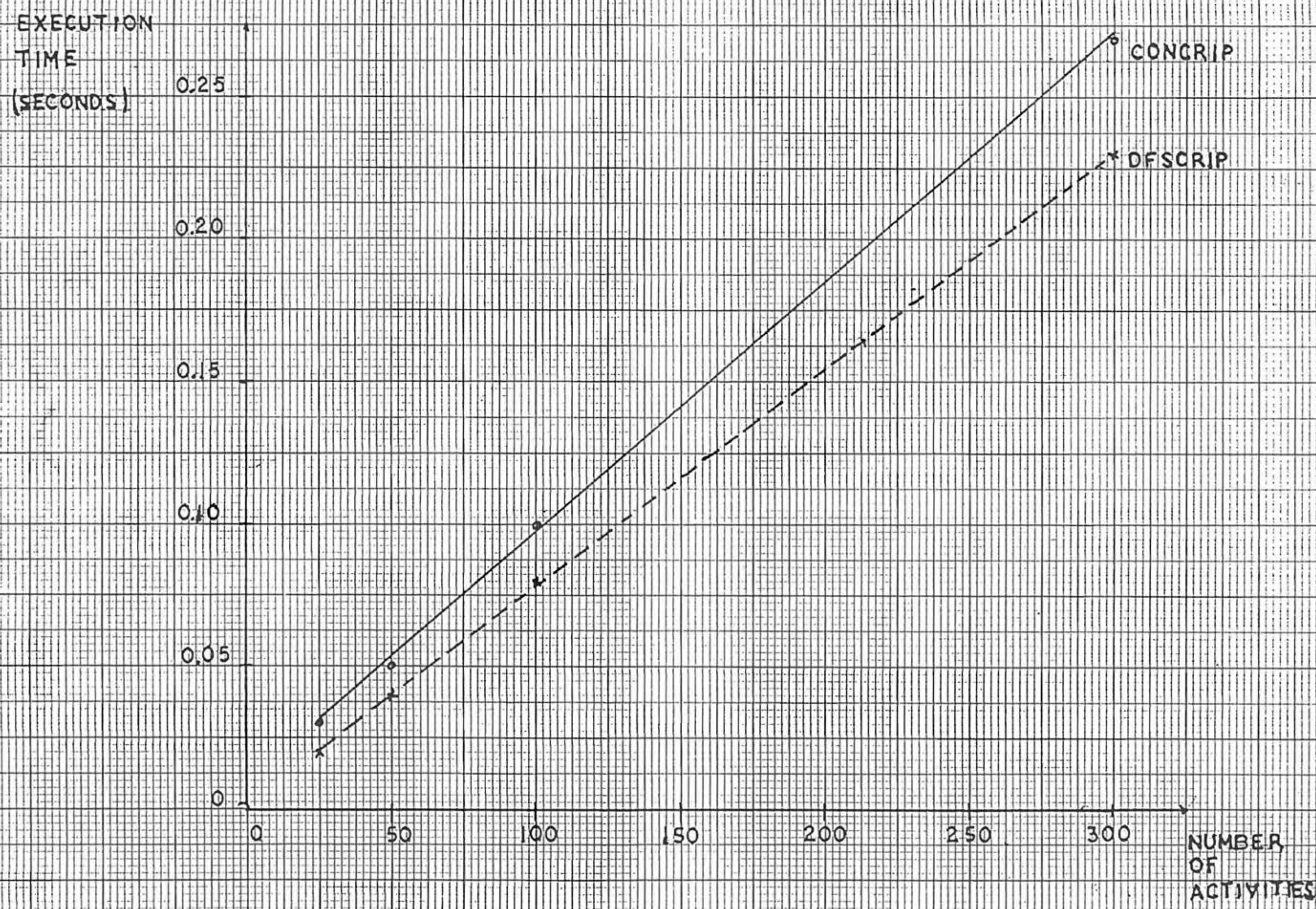



FIG. 6 EXECUTION TIMES FOR CPM ALGORITHMS

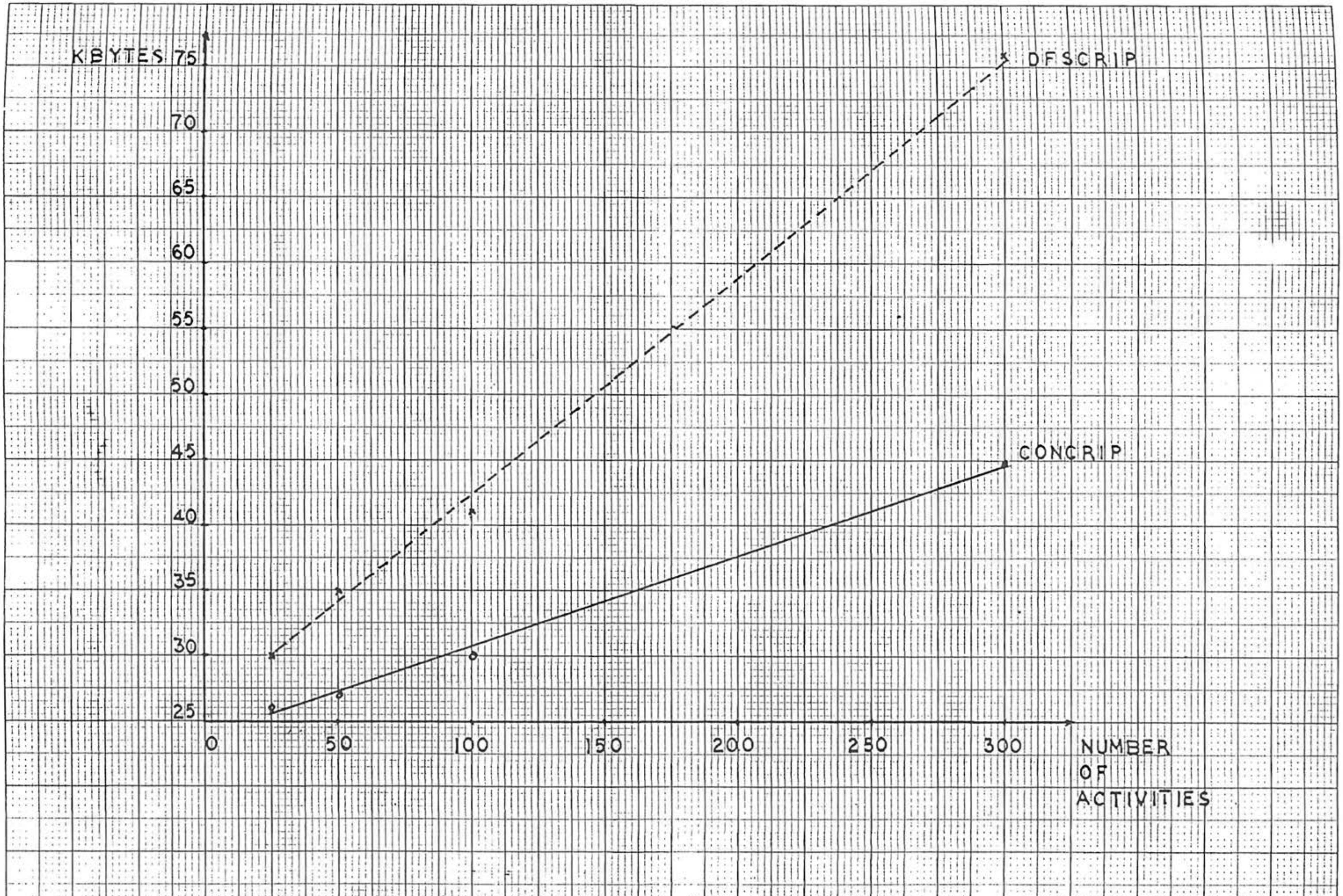



FIG. 7 STORAGE BYTES USED BY CPM ALGORITHMS

## 7.0 EVENT FLOAT

The concept of float can be extended to events as well and the idea of Event Float has been presented by Locks(10).An event is the boundary between a set of activities completed and a set of activities to be commenced.One might be interested in knowing the slack time available between these.Event Float is defined as the amount of slack available at an event when all preceeding and succeeding activities are carried out in the shortest possible time and the total project time remains unaffected.It is numerically evaluated at event x as $C - \{LET(x) + EET(x)\}$ ,where C is the critical path.Event Float can provide particularly useful interpretation if the event separates two distinctly different set of activities.All the events on the critical path have an Event Float equal to 0.

Event Float values,as well as the other floats for the example in section 8.0 are provided in the output printout in section 8.2.

## 8.0 Large Network Example

This section illustrates the usage of the computer program for a large network integrated from four modules.The supernetwork for this example is shown in Figure 8.

Module 1, illustrated in Figure 9, consists of 15 events and 25 activities. The duration of each activity ,in arbitrary time units, is shown alongside in the diagram.

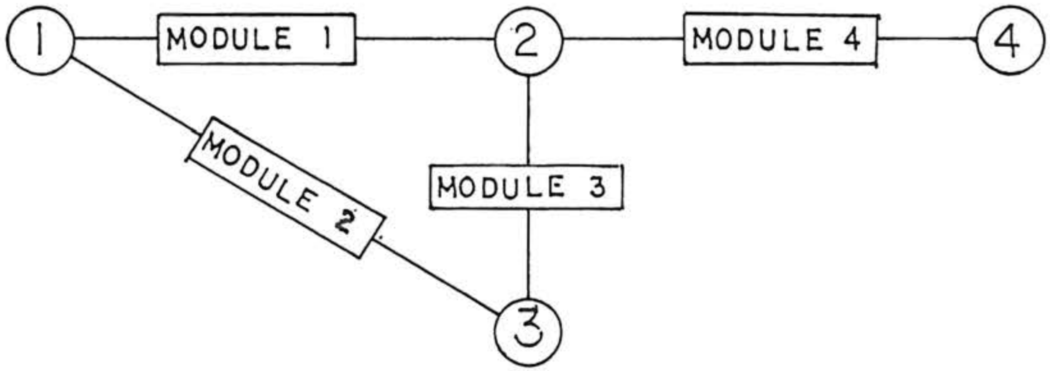
Module 2 consists of 25 events and 50 activities and is illustrated in Figure 10.

Module 3, shown in Figure 11, consists of 50 events and 100 activities.

Module 4, shown in Figure 12, consists of 160 events and 300 activities.

The entire problem needed .94 seconds of CPU time on the IBM 3081-D and the core required was 76 kBytes.

FIG 8 MODULE INTEGRATION EXAMPLE




SUPERNETWORK

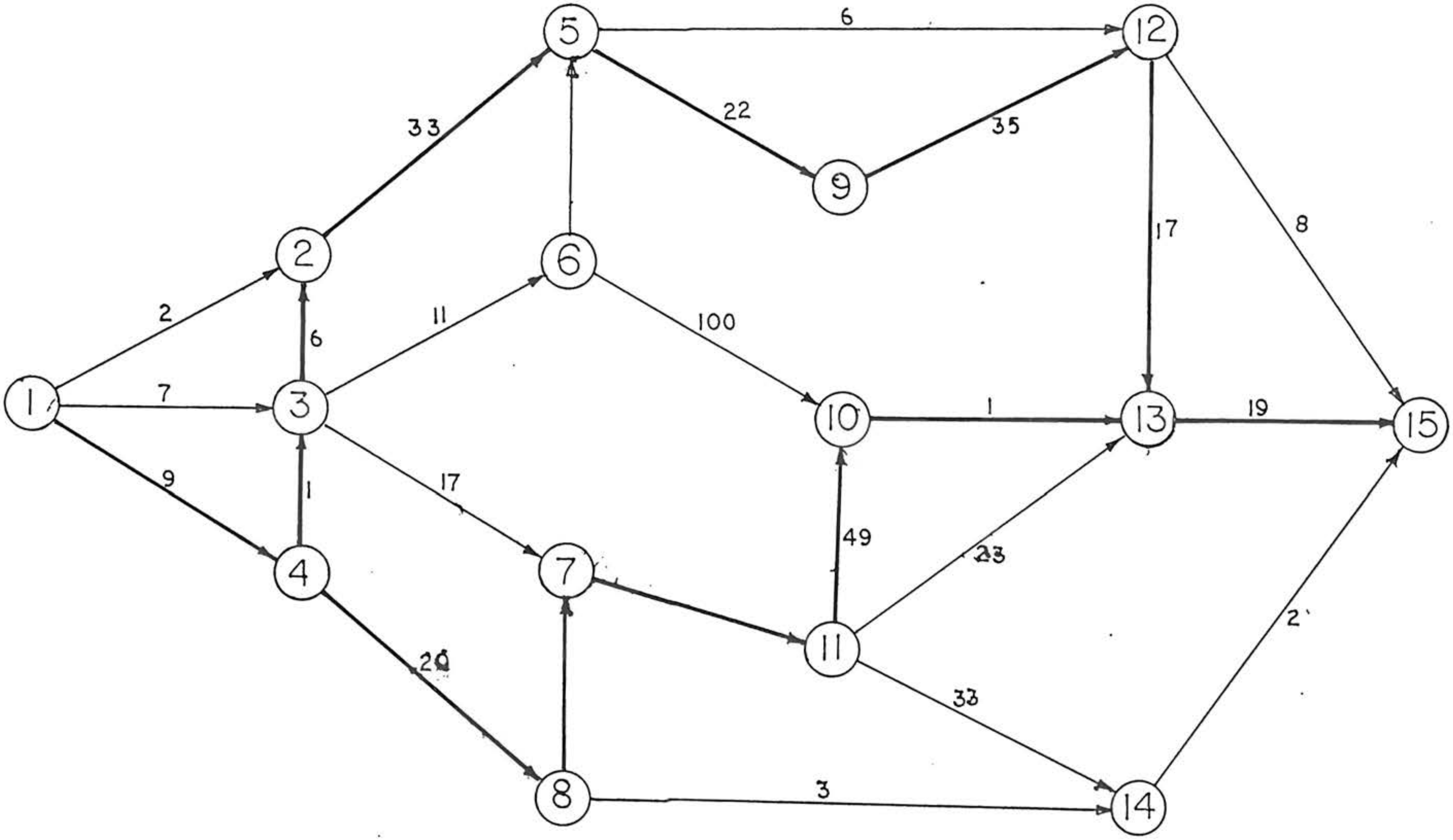



*FIG 9 LARGE NETWORK EXAMPLE*

MODULE I
25 ACTIVITIES

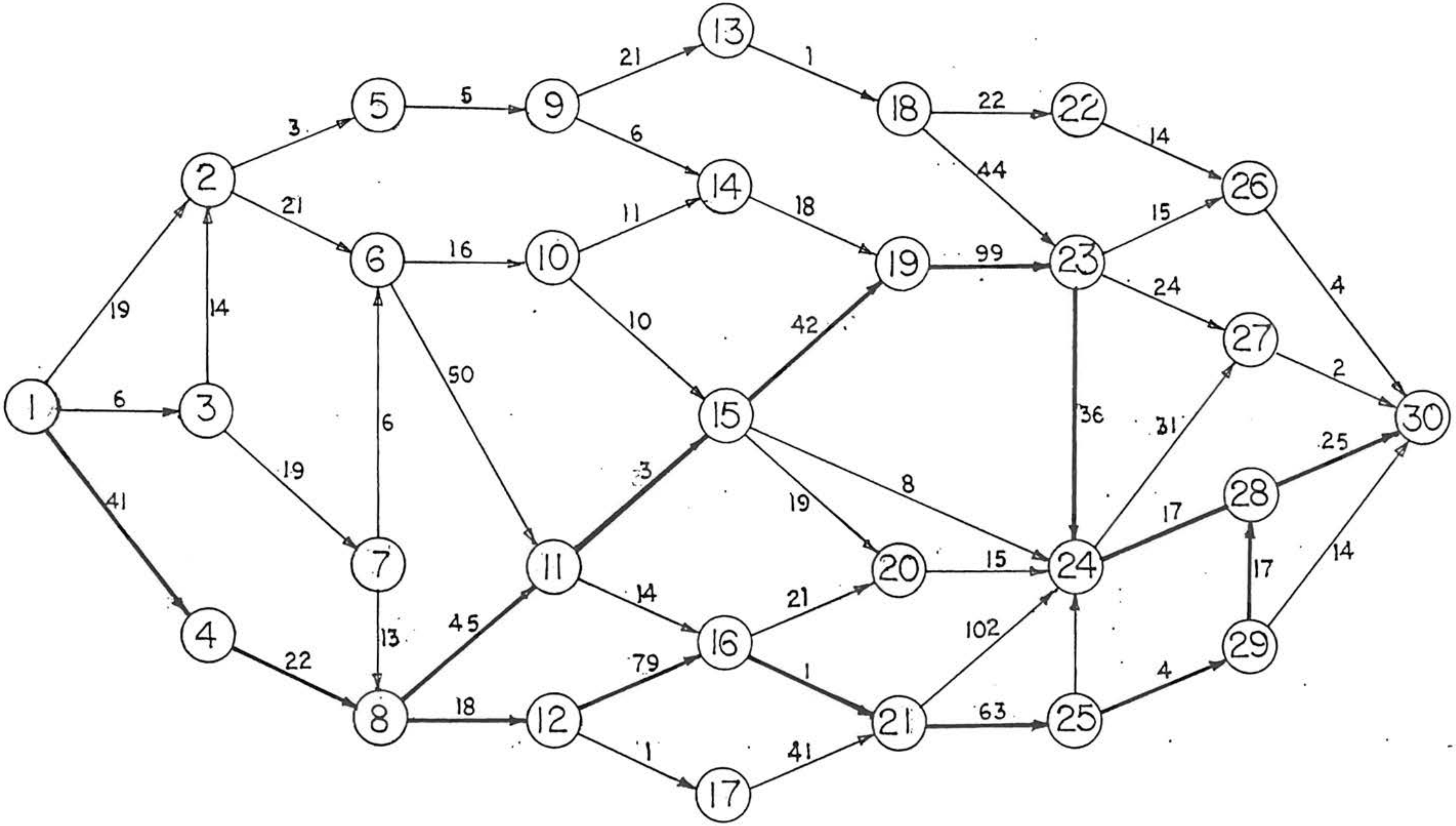



FIG 10 LARGE NETWORK EXAMPLE

MODULE 2
50 ACTIVITIES

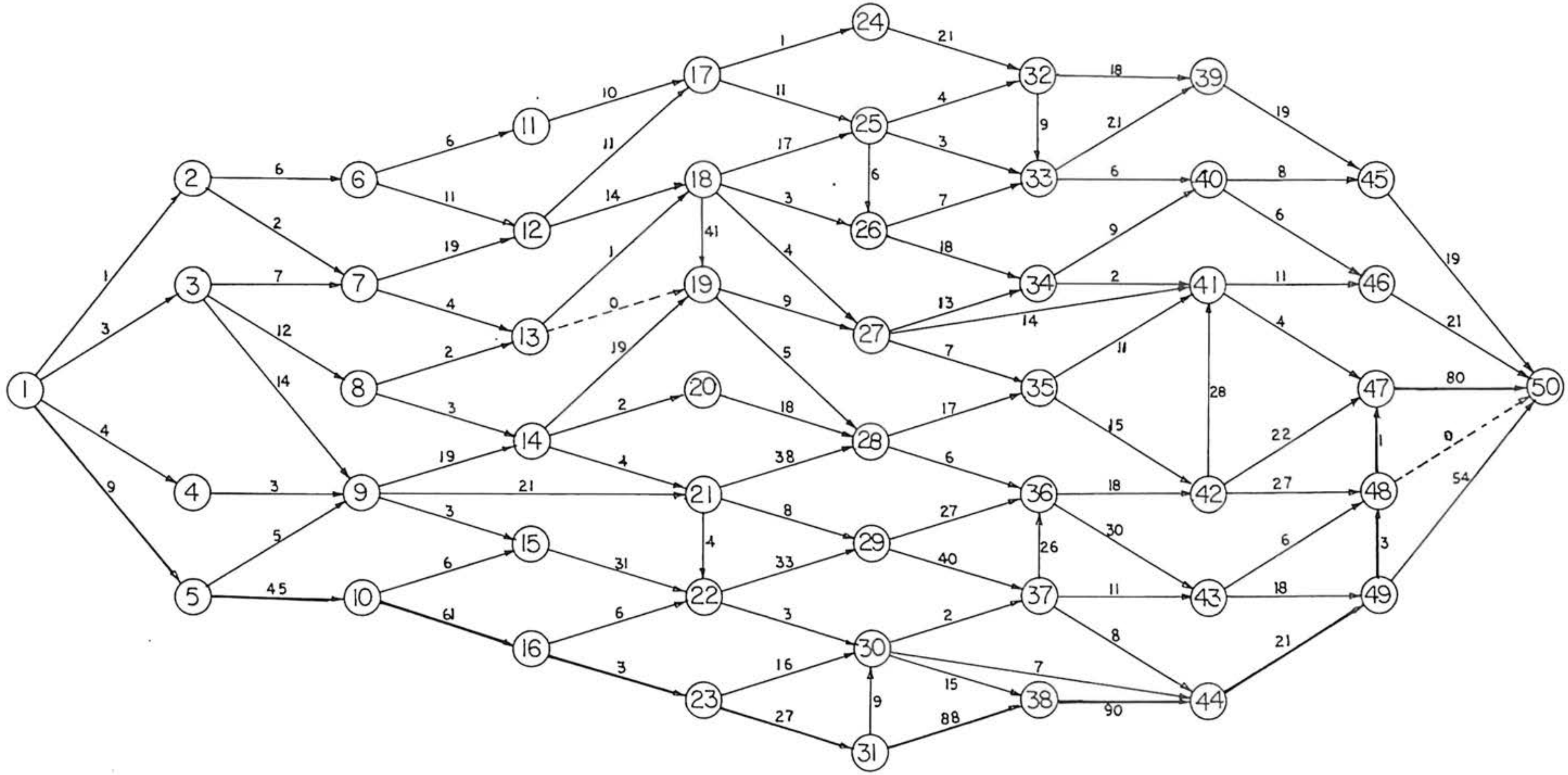
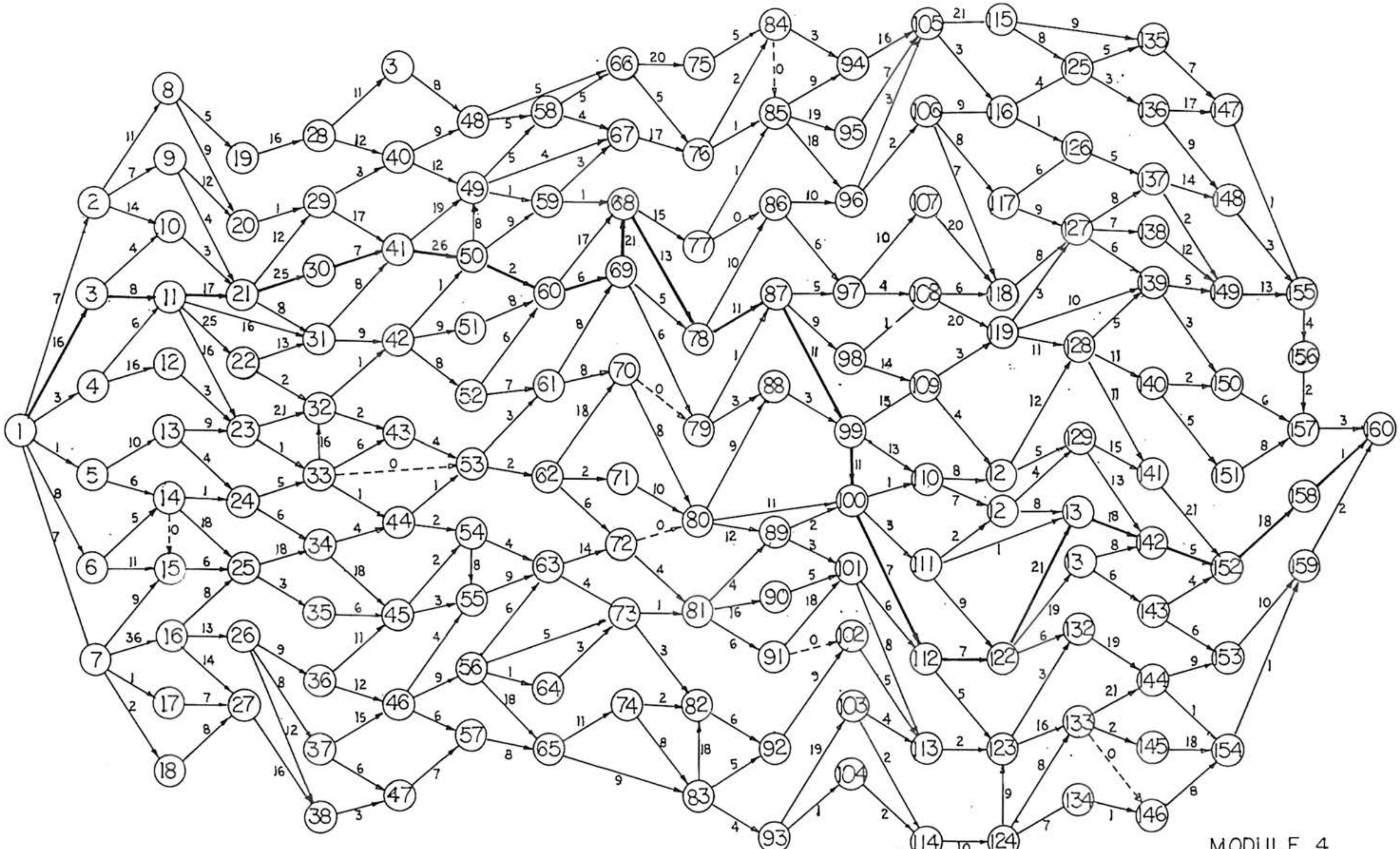


FIG 11 LARGE NETWORK EXAMPLE

MODULE 3
100 ACTIVITIES



MODULE 4
300 ACTIVITIES

*FIG 12 LARGE NETWORK EXAMPLE*

## 8.1 Input Data Listing

In the following pages a listing of the input data for the entire integration problem is supplied. Entries for the four modules are followed by the entries for the supernetwork in the last four lines.

```
'MODULE 1'  15
1  2  2  3  0  4  9  0
2  5  83 0
3  2  6  6  11  7  17  0
4  3  1  8  20  0
5  12  6  9  22  0
6  5  45  10  100  0
7  11  65  0
8  7  19  14  3  0
9  12  35  0
10  13  1  0
11  10  49  13  0  14  33  0
12  13  7  14  8  0
13  15  19  0
14  15  2  0
  'MODULE 2'  30
1  2  19  3  6  4  41  0
2  5  3  6  21  0
3  2  14  7  19  0
4  8  22  0
5  9  5  0
6  10  16  11  50  0
7  6  6  8  13  0
8  11  45  12  18  0
9  13  21  14  6  0
10  14  11  15  10  0
11  15  3  16  14  0
12  16  79  17  10  0
13  18  1  0
14  19  18  0
15  19  42  24  8  20  19  0
16  20  21  21  1  0
17  21  41  0
18  22  27  23  44  0
19  23  99  0
20  24  15  0
21  24  102  25  63  0
22  26  14  0
23  26  15  27  24  24  36  0
24  27  31  28  17  0
25  24  2  29  4  0
26  30  4  0
27  30  2  0
28  30  25  0
29  28  77  30  14  0
  'MODULE 3'  50
1 2 1 3 3 4 4 5 9 0
2 6 6 7 2 0
3 7 7 8 12 9 14 0
4 9 3 0
5 9 5 10 45 0
6 11 6 12 11  0
7 12 9 13 4 0
8 13 2 14 3 0
```

```
9 14 19 21 21 15 3 0
10 15 6 16 61 0
11 17 10 0
12 17 11 18 14 0
13 18 1 19 0 0
14 19 19 20 2 21 4 0
15 22 31 0
16 22 6 23 3 0
17 24 1 25 11 0
18 25 17 26 3 27 4 19 41 0
19 27 9 28 5 0
20 28 18 0
21 28 38 29 8 22 4 0
22 29 33 30 3 0
23 30 16 31 27 0
24 32 21 0
25 32 4 33 3 26 6 0
26 33 7 34 18 0
27 34 13 41 14 35 7 0
28 35 17 36 6 0
29 36 27 37 40 0
30 37 2 44 7 38 15 0
31 30 9 38 88 0
32 39 18 33 9 0
33 39 21 40 6 0
34 40 9 41 2 0
35 41 11 42 5 0
36 42 18 43 30 0
37 36 26 43 11 44 8 0
38 44 90 0
39 45 19 0
40 45 8 46 6 0
41 46 11 47 4 0
42 41 28 47 22 48 27 0
43 48 6 49 18 0
44 49 21 0
45 50 19 0
46 50 21 0
47 50 80 0
48 47 1 50 0 0
49 48 3 50 54 0
  'MODULE 4'   160
1  2  7  3  16  4  3  5  1  6  8  7  2  0
2  8  11  9  7  10  14  0
3  10  4  11  8  0
4  11  6  12  16  0
5  13  10  14  6  0
6  14  5  15  11  0
7  15  9  16  36 17  1  18  2  0
8  19  5  20  9  0
9  20  12  21  4  0
10  21  3  0
11  21  17  31  16  22  25  23  16  0
12  23  3  0
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 13 | 24 | 4 | 23 | 9 | 0 | | | | |
| 14 | 24 | 1 | 25 | 18 | 15 | 0 | 0 | | |
| 15 | 25 | 6 | 0 | | | | | | |
| 16 | 25 | 8 | 26 | 13 | 27 | 14 | 0 | | |
| 17 | 27 | 7 | 0 | | | | | | |
| 18 | 27 | 8 | 0 | | | | | | |
| 19 | 28 | 16 | 0 | | | | | | |
| 20 | 29 | 1 | 0 | | | | | | |
| 21 | 29 | 12 | 30 | 25 | 31 | 8 | 0 | | |
| 22 | 31 | 13 | 32 | 2 | 0 | | | | |
| 23 | 32 | 21 | 33 | 1 | 0 | | | | |
| 24 | 33 | 5 | 34 | 6 | 0 | | | | |
| 25 | 34 | 18 | 35 | 3 | 0 | | | | |
| 26 | 36 | 9 | 37 | 8 | 38 | 12 | 0 | | |
| 27 | 38 | 16 | 0 | | | | | | |
| 28 | 39 | 11 | 40 | 12 | 0 | | | | |
| 29 | 40 | 3 | 41 | 17 | 0 | | | | |
| 30 | 41 | 7 | 0 | | | | | | |
| 31 | 41 | 8 | 42 | 9 | 0 | | | | |
| 32 | 42 | 1 | 43 | 2 | 0 | | | | |
| 33 | 32 | 16 | 43 | 6 | 53 | 0 | 44 | 1 | 0 |
| 34 | 44 | 4 | 45 | 18 | 0 | | | | |
| 35 | 45 | 6 | 0 | | | | | | |
| 36 | 45 | 11 | 46 | 12 | 0 | | | | |
| 37 | 46 | 15 | 47 | 6 | 0 | | | | |
| 38 | 47 | 3 | 0 | | | | | | |
| 39 | 48 | 8 | 0 | | | | | | |
| 40 | 48 | 9 | 49 | 12 | 0 | | | | |
| 41 | 49 | 19 | 50 | 26 | 0 | | | | |
| 42 | 50 | 1 | 51 | 9 | 52 | 8 | 0 | | |
| 43 | 53 | 4 | 0 | | | | | | |
| 44 | 53 | 1 | 54 | 2 | 0 | | | | |
| 45 | 54 | 2 | 55 | 3 | 0 | | | | |
| 46 | 55 | 4 | 56 | 9 | 57 | 6 | 0 | | |
| 47 | 57 | 7 | 0 | | | | | | |
| 48 | 66 | 5 | 58 | 5 | 0 | | | | |
| 49 | 58 | 5 | 67 | 4 | 59 | 1 | 0 | | |
| 50 | 49 | 8 | 59 | 9 | 60 | 2 | 0 | | |
| 51 | 60 | 8 | 0 | | | | | | |
| 52 | 60 | 6 | 61 | 7 | 0 | | | | |
| 53 | 61 | 3 | 62 | 2 | 0 | | | | |
| 54 | 63 | 4 | 55 | 8 | 0 | | | | |
| 55 | 63 | 9 | 0 | | | | | | |
| 56 | 63 | 6 | 73 | 5 | 64 | 1 | 65 | 18 | 0 |
| 57 | 65 | 8 | 0 | | | | | | |
| 58 | 66 | 5 | 67 | 4 | 0 | | | | |
| 59 | 67 | 3 | 68 | 1 | 0 | | | | |
| 60 | 68 | 17 | 69 | 6 | 0 | | | | |
| 61 | 69 | 8 | 70 | 8 | 0 | | | | |
| 62 | 70 | 18 | 71 | 2 | 72 | 6 | 0 | | |
| 63 | 72 | 14 | 73 | 4 | 0 | | | | |
| 64 | 73 | 3 | 0 | | | | | | |
| 65 | 74 | 11 | 83 | 9 | 0 | | | | |
| 66 | 75 | 20 | 76 | 5 | 0 | | | | |

```
67   76   17   0
68   77   15   78   13   0
69   68   21   78   5   79   6   0
70   79   0   80   8   0
71   80   10   0
72   80   0   81   4   0
73   81   1   82   3   0
74   82   2   83   8   0
75   84 5 0
76   84   2   85   1   0
77   85   1   86   0   0
78   86   10   87   11   0
79   87   1   88   3   0
80   88   9   100   11   89   12   0
81   89   4   90   16   91   6   0
82   92   6   0
83   82   18   92   5   93   4   0
84   94   3   85   0   0
85   94   9   95   19   96   18   0
86   96   10   97   6 0
87   97   5   98   9 99   11   0
88   99   3   0
89   100 2 101   3   0
90   101   5   0
91   101   18   102   0   0
92   102   9   0
93   103   19   104   1   0
94   105   16   0
95   105   7   0
96   105   3   106   2   0
97   107   10   108   4   0
98   108   1   109   14   0
99   109   15   110 13   100   11   0
100   110   1   111   3   112   7   0
101   112   6   113   8   0
102   113   5   0
103   113   4   114   2   0
104   112   2 0
105   115   21   116   3   0
106   116   9   117   8   118   7   0
107   118   20   0
108   118   6   119   20   0
109   119   3   120   4   0
110   120   8   121   7   0
111   121   2   130   1   122 9 0
112   122 7   123   5   0
113   123   2   0
114   124   10 0
115   135   9   125   8   0
116   125   4   126   1   0
117   126   6   127   9   0
118   127   8   0
119   127   3   139   10   128   11   0
120   128   12   129   5   0
```

```
121   129   4   130   8   0
122   130   21   131   19   132   6   0
123   132   3   133   16   0
124   123   9   133   8   134   7   0
125   135   5   136   3   0
126   137   5   0
127   137   8   138   7   139   6   0
128   139   5   140   11   141   11   0
129   141   15   142   13   0
130   142   18   0
131   142   8   143   6   0
132   143   9   0
133   144   21   145   2   146   0   0
134   146   1   0
135   147   7   0
136   147   17   148   9   0
137   148   14   149   2   0
138   149   12 0
139   149   5   150   3   0
140   150   2   151   5   0
141   152   21   0
142   152   5 0
143   152   4   153   6   0
144   153   9   154   1   0
145   154   18   0
146   154   8   0
147   155 1   0
148   155 3   0
149   155 13   0
150   157   6   0
151   157   8   0
152   158   18   0
153   159   10   0
154   159   1   0
155   156   4   0
156   157   2   0
157   160   3   0
158   160   1   0
159   160   2   0
 'PROJECT'   4
1  2   'MODULE 1'  3  'MODULE 2'  0
2  4  'MODULE 4'  0
3  2  'MODULE 3'  0
//
```

## 8.2 Printout of Results

Printout for the complete problem is furnished in the following pages. The results for the supernetwork are given first.The critical path goes through events 1,3,2,4 in Figure 8.The total project time is 1009 arbitrary time units which is the sum of the critical path lengths for modules 2,3 and 4.The critical paths of the four individual modules can be read from the printout.

PROJECT

CRITICAL PATH LENGTH = 1009

CRITICAL PATH 1 3 2 4

| EVENT | EVENT FLOAT |
| --- | --- |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| 1 | ----- | 3 | 0 | 0 | 0 | 0 |
| 2 | ----- | 4 | 0 | 0 | 0 | 0 |
| 3 | ----- | 2 | 0 | 0 | 0 | 0 |

IN STMT 10 PROGRAM RETURNS FROM MAIN PROCEDURE.

CRITICAL PATH LENGTH = 182

| CRITICAL PATH | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 4 | 3 | 2 | 5 | 9 | 12 |
| 13 | 15 | 4 | 8 | 7 | 11 | 10 |
| 13 | 15 | | | | | |

| EVENT | EVENT FLOAT |
|---|---|
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 33 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |
| 11 | 0 |
| 12 | 0 |
| 13 | 0 |
| 14 | 16 |
| 15 | 0 |

| ACTIVITY | TOTAL FLOAT | FREE FLOAT | INDEP. FLOAT | SAFETY FLOAT |
|---|---|---|---|---|
| 1 ----- 2 | 14 | 14 | 14 | 14 |
| 1 ----- 4 | 0 | 0 | 0 | 0 |
| 2 ----- 5 | 0 | 0 | 0 | 0 |
| 3 ----- 2 | 0 | 0 | 0 | 0 |
| 3 ----- 6 | 33 | 0 | 0 | 33 |
| 3 ----- 7 | 21 | 21 | 21 | 21 |
| 4 ----- 3 | 0 | 0 | 0 | 0 |
| 4 ----- 8 | 0 | 0 | 0 | 0 |
| 5 ----- 12 | 51 | 51 | 51 | 51 |
| 5 ----- 9 | 0 | 0 | 0 | 0 |
| 6 ----- 5 | 33 | 33 | 0 | 0 |
| 6 ----- 10 | 41 | 41 | 8 | 8 |
| 7 ----- 11 | 0 | 0 | 0 | 0 |
| 8 ----- 7 | 0 | 0 | 0 | 0 |
| 8 ----- 14 | 148 | 132 | 132 | 148 |
| 9 ----- 12 | 0 | 0 | 0 | 0 |
| 10 ----- 13 | 0 | 0 | 0 | 0 |
| 11 ----- 10 | 0 | 0 | 0 | 0 |
| 11 ----- 14 | 34 | 18 | 18 | 34 |
| 12 ----- 13 | 0 | 0 | 0 | 0 |
| 12 ----- 14 | 16 | 0 | 0 | 16 |
| 13 ----- 15 | 0 | 0 | 0 | 0 |
| 14 ----- 15 | 16 | 16 | 0 | 0 |

CRITICAL PATH LENGTH = 330

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| CRITICAL PATH | 1 | 4 | 8 | 11 | 15 | 19 | 23 |
| | 24 | 28 | 30 | 8 | 12 | 16 | 21 |
| | 25 | 29 | 28 | 30 | | | |

| EVENT | EVENT FLOAT |
|---|---|
| 1 | 0 |
| 2 | 17 |
| 3 | 17 |
| 4 | 0 |
| 5 | 101 |
| 6 | 17 |
| 7 | 25 |
| 8 | 0 |
| 9 | 101 |
| 10 | 44 |
| 11 | 0 |
| 12 | 0 |
| 13 | 158 |
| 14 | 67 |
| 15 | 0 |
| 16 | 0 |
| 17 | 29 |
| 18 | 158 |
| 19 | 0 |
| 20 | 92 |
| 21 | 0 |
| 22 | 235 |
| 23 | 0 |
| 24 | 0 |
| 25 | 0 |
| 26 | 59 |
| 27 | 9 |
| 28 | 0 |
| 29 | 0 |
| 30 | 0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | ----- | 2 | 18 | 1 | 1 | 18 |
| 1 | ----- | 3 | 17 | 0 | 0 | 17 |
| 1 | ----- | 4 | 0 | 0 | 0 | 0 |
| 2 | ----- | 5 | 101 | 0 | 0 | 84 |
| 2 | ----- | 6 | 17 | 0 | 0 | 0 |
| 3 | ----- | 2 | 17 | 0 | 0 | 0 |
| 3 | ----- | 7 | 25 | 0 | 0 | 8 |
| 4 | ----- | 8 | 0 | 0 | 0 | 0 |
| 5 | ----- | 9 | 101 | 0 | 0 | 0 |
| 6 | ----- | 10 | 44 | 0 | 0 | 27 |
| 6 | ----- | 11 | 17 | 17 | 0 | 0 |
| 7 | ----- | 6 | 27 | 10 | 0 | 2 |
| 7 | ----- | 8 | 25 | 25 | 0 | 0 |
| 8 | ----- | 11 | 0 | 0 | 0 | 0 |
| 8 | ----- | 12 | 0 | 0 | 0 | 0 |
| 9 | ----- | 13 | 158 | 0 | 0 | 57 |
| 9 | ----- | 14 | 101 | 34 | 0 | 0 |
| 10 | ----- | 14 | 67 | 0 | 0 | 23 |
| 10 | ----- | 15 | 44 | 44 | 0 | 0 |
| 11 | ----- | 15 | 0 | 0 | 0 | 0 |
| 11 | ----- | 16 | 38 | 38 | 38 | 38 |
| 12 | ----- | 16 | 0 | 0 | 0 | 0 |
| 12 | ----- | 17 | 29 | 0 | 0 | 29 |
| 13 | ----- | 18 | 158 | 0 | 0 | 0 |
| 14 | ----- | 19 | 67 | 67 | 0 | 0 |
| 15 | ----- | 19 | 0 | 0 | 0 | 0 |
| 15 | ----- | 24 | 169 | 169 | 169 | 169 |
| 15 | ----- | 20 | 143 | 51 | 51 | 143 |
| 16 | ----- | 20 | 92 | 0 | 0 | 92 |
| 16 | ----- | 21 | 0 | 0 | 0 | 0 |
| 17 | ----- | 21 | 29 | 29 | 0 | 0 |
| 18 | ----- | 22 | 235 | 0 | 0 | 77 |
| 18 | ----- | 23 | 158 | 158 | 0 | 0 |
| 19 | ----- | 23 | 0 | 0 | 0 | 0 |
| 20 | ----- | 24 | 92 | 92 | 0 | 0 |
| 21 | ----- | 24 | 25 | 25 | 25 | 25 |
| 21 | ----- | 25 | 0 | 0 | 0 | 0 |
| 22 | ----- | 26 | 235 | 176 | 0 | 0 |
| 23 | ----- | 26 | 59 | 0 | 0 | 59 |
| 23 | ----- | 27 | 52 | 43 | 43 | 52 |
| 23 | ----- | 24 | 0 | 0 | 0 | 0 |
| 24 | ----- | 27 | 9 | 0 | 0 | 9 |
| 24 | ----- | 28 | 0 | 0 | 0 | 0 |
| 25 | ----- | 24 | 62 | 62 | 62 | 62 |
| 25 | ----- | 29 | 0 | 0 | 0 | 0 |
| 26 | ----- | 30 | 59 | 59 | 0 | 0 |
| 27 | ----- | 30 | 9 | 9 | 0 | 0 |
| 28 | ----- | 30 | 0 | 0 | 0 | 0 |
| 29 | ----- | 28 | 0 | 0 | 0 | 0 |
| 29 | ----- | 30 | 88 | 88 | 88 | 88 |

MODULE 3

CRITICAL PATH LENGTH = 428

CRITICAL PATH 1 5 10 16 23 31 38
44 49 48 47 50

| EVENT | EVENT FLOAT |
|---|---|
| 1 | 0 |
| 2 | 212 |
| 3 | 146 |
| 4 | 156 |
| 5 | 0 |
| 6 | 212 |
| 7 | 211 |
| 8 | 171 |
| 9 | 146 |
| 10 | 0 |
| 11 | 284 |
| 12 | 211 |
| 13 | 226 |
| 14 | 153 |
| 15 | 106 |
| 16 | 0 |
| 17 | 277 |
| 18 | 211 |
| 19 | 211 |
| 20 | 234 |
| 21 | 153 |
| 22 | 76 |
| 23 | 0 |
| 24 | 308 |
| 25 | 268 |
| 26 | 268 |
| 27 | 221 |
| 28 | 211 |
| 29 | 76 |
| 30 | 64 |
| 31 | 0 |
| 32 | 306 |
| 33 | 306 |
| 34 | 246 |
| 35 | 215 |
| 36 | 76 |
| 37 | 76 |
| 38 | 0 |
| 39 | 306 |
| 40 | 296 |
| 41 | 78 |
| 42 | 78 |
| 43 | 76 |
| 44 | 0 |
| 45 | 296 |
| 46 | 130 |
| 47 | 0 |

| | |
|---|---|
| 49 | 0 |
| 50 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| 1 | ----- | 2 | 212 | 0 | 0 | 212 |
| 1 | ----- | 3 | 146 | 0 | 0 | 146 |
| 1 | ----- | 4 | 156 | 0 | 0 | 156 |
| 1 | ----- | 5 | 0 | 0 | 0 | 0 |
| 2 | ----- | 6 | 212 | 0 | 0 | 0 |
| 2 | ----- | 7 | 218 | 7 | 0 | 6 |
| 3 | ----- | 7 | 211 | 0 | 0 | 65 |
| 3 | ----- | 8 | 171 | 0 | 0 | 25 |
| 3 | ----- | 9 | 146 | 0 | 0 | 0 |
| 4 | ----- | 9 | 156 | 10 | 0 | 0 |
| 5 | ----- | 9 | 149 | 3 | 3 | 149 |
| 5 | ----- | 10 | 0 | 0 | 0 | 0 |
| 6 | ----- | 11 | 284 | 0 | 0 | 72 |
| 6 | ----- | 12 | 212 | 1 | 0 | 0 |
| 7 | ----- | 12 | 211 | 0 | 0 | 0 |
| 7 | ----- | 13 | 229 | 3 | 0 | 18 |
| 8 | ----- | 13 | 226 | 0 | 0 | 55 |
| 8 | ----- | 14 | 171 | 18 | 0 | 0 |
| 9 | ----- | 14 | 153 | 0 | 0 | 7 |
| 9 | ----- | 21 | 155 | 2 | 0 | 9 |
| 9 | ----- | 15 | 146 | 40 | 0 | 0 |
| 10 | ----- | 15 | 106 | 0 | 0 | 106 |
| 10 | ----- | 16 | 0 | 0 | 0 | 0 |
| 11 | ----- | 17 | 284 | 7 | 0 | 0 |
| 12 | ----- | 17 | 277 | 0 | 0 | 66 |
| 12 | ----- | 18 | 211 | 0 | 0 | 0 |
| 13 | ----- | 18 | 226 | 15 | 0 | 0 |
| 14 | ----- | 19 | 230 | 19 | 0 | 77 |
| 14 | ----- | 20 | 234 | 0 | 0 | 81 |
| 14 | ----- | 21 | 153 | 0 | 0 | 0 |
| 15 | ----- | 22 | 106 | 30 | 0 | 0 |
| 16 | ----- | 22 | 76 | 0 | 0 | 76 |
| 16 | ----- | 23 | 0 | 0 | 0 | 0 |
| 17 | ----- | 24 | 308 | 0 | 0 | 31 |
| 17 | ----- | 25 | 277 | 9 | 0 | 0 |
| 18 | ----- | 25 | 268 | 0 | 0 | 57 |
| 18 | ----- | 26 | 288 | 20 | 0 | 77 |
| 18 | ----- | 27 | 267 | 46 | 0 | 56 |
| 18 | ----- | 19 | 211 | 0 | 0 | 0 |
| 19 | ----- | 27 | 221 | 0 | 0 | 10 |
| 19 | ----- | 28 | 211 | 0 | 0 | 0 |
| 20 | ----- | 28 | 234 | 23 | 0 | 0 |
| 21 | ----- | 28 | 212 | 1 | 0 | 59 |
| 21 | ----- | 29 | 182 | 106 | 0 | 29 |
| 21 | ----- | 22 | 153 | 77 | 0 | 0 |
| 22 | ----- | 29 | 76 | 0 | 0 | 0 |
| 22 | ----- | 30 | 94 | 30 | 0 | 18 |
| 23 | ----- | 30 | 84 | 20 | 20 | 84 |
| 23 | ----- | 31 | 0 | 0 | 0 | 0 |
| 24 | ----- | 32 | 308 | 2 | 0 | 0 |
| 25 | ----- | 32 | 306 | 0 | 0 | 38 |
| 25 | ----- | 33 | 316 | 10 | 0 | 48 |
| 25 | ----- | 26 | 268 | 0 | 0 | 0 |
| 26 | ----- | 33 | 306 | 0 | 0 | 38 |
| 26 | ----- | 34 | 268 | 22 | 0 | 0 |
| 27 | ----- | 34 | 246 | 0 | 0 | 25 |
| 27 | ----- | 41 | 247 | 169 | 0 | 26 |
| 27 | ----- | 35 | 221 | 6 | 0 | 0 |
| 28 | ----- | 35 | 215 | 0 | 0 | 4 |

| | | | | | |
|---|---|---|---|---|---|
| 28 | ----- 36 | 211 | 135 | 0 | 0 |
| 29 | ----- 36 | 115 | 39 | 0 | 39 |
| 29 | ----- 37 | 76 | 0 | 0 | 0 |
| 30 | ----- 37 | 114 | 38 | 0 | 50 |
| 30 | ----- 44 | 162 | 162 | 98 | 98 |
| 30 | ----- 38 | 64 | 64 | 0 | 0 |
| 31 | ----- 30 | 64 | 0 | 0 | 64 |
| 31 | ----- 38 | 0 | 0 | 0 | 0 |
| 32 | ----- 39 | 318 | 12 | 0 | 12 |
| 32 | ----- 33 | 306 | 0 | 0 | 0 |
| 33 | ----- 39 | 306 | 0 | 0 | 0 |
| 33 | ----- 40 | 332 | 36 | 0 | 26 |
| 34 | ----- 40 | 296 | 0 | 0 | 50 |
| 34 | ----- 41 | 246 | 168 | 0 | 0 |
| 35 | ----- 41 | 237 | 159 | 0 | 22 |
| 35 | ----- 42 | 215 | 137 | 0 | 0 |
| 36 | ----- 42 | 78 | 0 | 0 | 2 |
| 36 | ----- 43 | 76 | 0 | 0 | 0 |
| 37 | ----- 36 | 76 | 0 | 0 | 0 |
| 37 | ----- 43 | 121 | 45 | 0 | 45 |
| 37 | ----- 44 | 121 | 121 | 45 | 45 |
| 38 | ----- 44 | 0 | 0 | 0 | 0 |
| 39 | ----- 45 | 306 | 10 | 0 | 0 |
| 40 | ----- 45 | 296 | 0 | 0 | 0 |
| 40 | ----- 46 | 296 | 166 | 0 | 0 |
| 41 | ----- 46 | 130 | 0 | 0 | 52 |
| 41 | ----- 47 | 78 | 78 | 0 | 0 |
| 42 | ----- 41 | 78 | 0 | 0 | 0 |
| 42 | ----- 47 | 88 | 88 | 10 | 10 |
| 42 | ----- 48 | 82 | 82 | 4 | 4 |
| 43 | ----- 48 | 91 | 91 | 15 | 15 |
| 43 | ----- 49 | 76 | 76 | 0 | 0 |
| 44 | ----- 49 | 0 | 0 | 0 | 0 |
| 45 | ----- 50 | 296 | 296 | 0 | 0 |
| 46 | ----- 50 | 130 | 130 | 0 | 0 |
| 47 | ----- 50 | 0 | 0 | 0 | 0 |
| 48 | ----- 47 | 0 | 0 | 0 | 0 |
| 49 | ----- 48 | 0 | 0 | 0 | 0 |
| 49 | ----- 50 | 30 | 30 | 30 | 30 |

MODULE 4

CRITICAL PATH LENGTH = 251

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| CRITICAL PATH | 1 | 3 | 11 | 21 | 30 | 41 | 50 |
| | 60 | 69 | 68 | 78 | 87 | 99 | 100 |
| | 112 | 122 | 130 | 142 | 152 | 158 | 160 |

| EVENT | EVENT FLOAT |
|---|---|
| 1 | 0 |
| 2 | 17 |
| 3 | 0 |
| 4 | 15 |
| 5 | 42 |
| 6 | 42 |
| 7 | 27 |
| 8 | 28 |
| 9 | 23 |
| 10 | 17 |
| 11 | 0 |
| 12 | 40 |
| 13 | 42 |
| 14 | 42 |
| 15 | 48 |
| 16 | 27 |
| 17 | 86 |
| 18 | 84 |
| 19 | 63 |
| 20 | 28 |
| 21 | 0 |
| 22 | 3 |
| 23 | 22 |
| 24 | 47 |
| 25 | 27 |
| 26 | 29 |
| 27 | 44 |
| 28 | 63 |
| 29 | 3 |
| 30 | 0 |
| 31 | 3 |
| 32 | 22 |
| 33 | 26 |
| 34 | 27 |
| 35 | 59 |
| 36 | 31 |
| 37 | 29 |
| 38 | 44 |
| 39 | 73 |
| 40 | 58 |
| 41 | 0 |
| 42 | 13 |
| 43 | 29 |
| 44 | 27 |
| 45 | 32 |
| 46 | 29 |

| | |
|---|---|
| 48 | 66 |
| 49 | 19 |
| 50 | 0 |
| 51 | 13 |
| 52 | 13 |
| 53 | 27 |
| 54 | 32 |
| 55 | 32 |
| 56 | 29 |
| 57 | 42 |
| 58 | 24 |
| 59 | 19 |
| 60 | 0 |
| 61 | 13 |
| 62 | 54 |
| 63 | 32 |
| 64 | 63 |
| 65 | 29 |
| 66 | 24 |
| 67 | 31 |
| 68 | 0 |
| 69 | 0 |
| 70 | 49 |
| 71 | 68 |
| 72 | 32 |
| 73 | 45 |
| 74 | 29 |
| 75 | 24 |
| 76 | 31 |
| 77 | 16 |
| 78 | 0 |
| 79 | 38 |
| 80 | 36 |
| 81 | 32 |
| 82 | 33 |
| 83 | 29 |
| 84 | 24 |
| 85 | 22 |
| 86 | 8 |
| 87 | 0 |
| 88 | 36 |
| 89 | 45 |
| 90 | 35 |
| 91 | 32 |
| 92 | 33 |
| 93 | 29 |
| 94 | 23 |
| 95 | 22 |
| 96 | 27 |
| 97 | 8 |
| 98 | 7 |
| 99 | 0 |
| 100 | 0 |
| 101 | 32 |
| 102 | 33 |
| 103 | 29 |
| 104 | 54 |
| 105 | 22 |
| 106 | 29 |

| | |
|---|---|
| 107 | 15 |
| 108 | 7 |
| 109 | 6 |
| 110 | 4 |
| 111 | 2 |
| 112 | 0 |
| 113 | 33 |
| 114 | 29 |
| 115 | 22 |
| 116 | 44 |
| 117 | 29 |
| 118 | 15 |
| 119 | 7 |
| 120 | 4 |
| 121 | 9 |
| 122 | 0 |
| 123 | 7 |
| 124 | 29 |
| 125 | 22 |
| 126 | 42 |
| 127 | 15 |
| 128 | 4 |
| 129 | 7 |
| 130 | 0 |
| 131 | 12 |
| 132 | 25 |
| 133 | 7 |
| 134 | 77 |
| 135 | 30 |
| 136 | 22 |
| 137 | 22 |
| 138 | 15 |
| 139 | 23 |
| 140 | 28 |
| 141 | 4 |
| 142 | 0 |
| 143 | 15 |
| 144 | 7 |
| 145 | 26 |
| 146 | 38 |
| 147 | 22 |
| 148 | 22 |
| 149 | 15 |
| 150 | 33 |
| 151 | 28 |
| 152 | 0 |
| 153 | 7 |
| 154 | 24 |
| 155 | 15 |
| 156 | 15 |
| 157 | 15 |
| 158 | 0 |
| 159 | 7 |
| 160 | 0 |

| | | | | |
|---|---|---|---|---|
| 1 ----- 2 | 17 | 0 | 0 | 17 |
| 1 ----- 3 | 0 | 0 | 0 | 0 |
| 1 ----- 4 | 15 | 0 | 0 | 15 |
| 1 ----- 5 | 42 | 0 | 0 | 42 |
| 1 ----- 6 | 42 | 0 | 0 | 42 |
| 1 ----- 7 | 27 | 0 | 0 | 27 |
| 2 ----- 8 | 28 | 0 | 0 | 11 |
| 2 ----- 9 | 23 | 0 | 0 | 6 |
| 2 ----- 10 | 17 | 0 | 0 | 0 |
| 3 ----- 10 | 18 | 1 | 1 | 18 |
| 3 ----- 11 | 0 | 0 | 0 | 0 |
| 4 ----- 11 | 15 | 15 | 0 | 0 |
| 4 ----- 12 | 40 | 0 | 0 | 25 |
| 5 ----- 13 | 42 | 0 | 0 | 0 |
| 5 ----- 14 | 48 | 6 | 0 | 6 |
| 6 ----- 14 | 42 | 0 | 0 | 0 |
| 6 ----- 15 | 48 | 0 | 0 | 6 |
| 7 ----- 15 | 56 | 8 | 0 | 29 |
| 7 ----- 16 | 27 | 0 | 0 | 0 |
| 7 ----- 17 | 86 | 0 | 0 | 59 |
| 7 ----- 18 | 84 | 0 | 0 | 57 |
| 8 ----- 19 | 63 | 0 | 0 | 35 |
| 8 ----- 20 | 28 | 0 | 0 | 0 |
| 9 ----- 20 | 29 | 1 | 0 | 6 |
| 9 ----- 21 | 23 | 23 | 0 | 0 |
| 10 ----- 21 | 17 | 17 | 0 | 0 |
| 11 ----- 21 | 0 | 0 | 0 | 0 |
| 11 ----- 31 | 25 | 22 | 22 | 25 |
| 11 ----- 22 | 3 | 0 | 0 | 3 |
| 11 ----- 23 | 22 | 0 | 0 | 22 |
| 12 ----- 23 | 40 | 18 | 0 | 0 |
| 13 ----- 24 | 47 | 0 | 0 | 5 |
| 13 ----- 23 | 42 | 20 | 0 | 0 |
| 14 ----- 24 | 48 | 1 | 0 | 6 |
| 14 ----- 25 | 42 | 15 | 0 | 0 |
| 15 ----- 25 | 48 | 21 | 0 | 0 |
| 16 ----- 25 | 27 | 0 | 0 | 0 |
| 16 ----- 26 | 29 | 0 | 0 | 2 |
| 16 ----- 27 | 44 | 0 | 0 | 17 |
| 17 ----- 27 | 86 | 42 | 0 | 0 |
| 18 ----- 27 | 84 | 40 | 0 | 0 |
| 19 ----- 28 | 63 | 0 | 0 | 0 |
| 20 ----- 29 | 28 | 25 | 0 | 0 |
| 21 ----- 29 | 3 | 0 | 0 | 3 |
| 21 ----- 30 | 0 | 0 | 0 | 0 |
| 21 ----- 31 | 16 | 13 | 13 | 16 |
| 22 ----- 31 | 3 | 0 | 0 | 0 |
| 22 ----- 32 | 32 | 10 | 7 | 29 |
| 23 ----- 32 | 22 | 0 | 0 | 0 |
| 23 ----- 33 | 26 | 0 | 0 | 4 |
| 24 ----- 33 | 47 | 21 | 0 | 0 |
| 24 ----- 34 | 70 | 43 | 0 | 23 |
| 25 ----- 34 | 27 | 0 | 0 | 0 |
| 25 ----- 35 | 59 | 0 | 0 | 32 |
| 26 ----- 36 | 31 | 0 | 0 | 2 |
| 26 ----- 37 | 29 | 0 | 0 | 0 |
| 26 ----- 38 | 49 | 5 | 0 | 20 |
| 27 ----- 38 | 44 | 0 | 0 | 0 |
| 28 ----- 39 | 73 | 0 | 0 | 10 |

| | | | | | |
|---|---|---|---|---|---|
| 28 | ----- 40 | 63 | 5 | 0 | 0 |
| 29 | ----- 40 | 58 | 0 | 0 | 55 |
| 29 | ----- 41 | 3 | 3 | 0 | 0 |
| 30 | ----- 41 | 0 | 0 | 0 | 0 |
| 31 | ----- 41 | 3 | 3 | 0 | 0 |
| 31 | ----- 42 | 13 | 0 | 0 | 10 |
| 32 | ----- 42 | 22 | 9 | 0 | 0 |
| 32 | ----- 43 | 29 | 0 | 0 | 7 |
| 33 | ----- 32 | 26 | 4 | 0 | 0 |
| 33 | ----- 43 | 45 | 16 | 0 | 19 |
| 33 | ----- 44 | 53 | 26 | 0 | 27 |
| 34 | ----- 44 | 27 | 0 | 0 | 0 |
| 34 | ----- 45 | 32 | 0 | 0 | 5 |
| 35 | ----- 45 | 59 | 27 | 0 | 0 |
| 36 | ----- 45 | 43 | 11 | 0 | 12 |
| 36 | ----- 46 | 31 | 2 | 0 | 0 |
| 37 | ----- 46 | 29 | 0 | 0 | 0 |
| 37 | ----- 47 | 50 | 6 | 0 | 21 |
| 38 | ----- 47 | 44 | 0 | 0 | 0 |
| 39 | ----- 48 | 73 | 7 | 0 | 0 |
| 40 | ----- 48 | 66 | 0 | 0 | 8 |
| 40 | ----- 49 | 58 | 39 | 0 | 0 |
| 41 | ----- 49 | 34 | 15 | 15 | 34 |
| 41 | ----- 50 | 0 | 0 | 0 | 0 |
| 42 | ----- 50 | 27 | 27 | 14 | 14 |
| 42 | ----- 51 | 13 | 0 | 0 | 0 |
| 42 | ----- 52 | 13 | 0 | 0 | 0 |
| 43 | ----- 53 | 29 | 2 | 0 | 0 |
| 44 | ----- 53 | 27 | 0 | 0 | 0 |
| 44 | ----- 54 | 46 | 14 | 0 | 19 |
| 45 | ----- 54 | 32 | 0 | 0 | 0 |
| 45 | ----- 55 | 39 | 7 | 0 | 7 |
| 46 | ----- 55 | 46 | 14 | 0 | 17 |
| 46 | ----- 56 | 29 | 0 | 0 | 0 |
| 46 | ----- 57 | 42 | 0 | 0 | 13 |
| 47 | ----- 57 | 44 | 2 | 0 | 0 |
| 48 | ----- 66 | 71 | 47 | 0 | 5 |
| 48 | ----- 58 | 66 | 42 | 0 | 0 |
| 49 | ----- 58 | 24 | 0 | 0 | 5 |
| 49 | ----- 67 | 36 | 5 | 0 | 17 |
| 49 | ----- 59 | 19 | 0 | 0 | 0 |
| 50 | ----- 49 | 19 | 0 | 0 | 19 |
| 50 | ----- 59 | 19 | 0 | 0 | 19 |
| 50 | ----- 60 | 0 | 0 | 0 | 0 |
| 51 | ----- 60 | 13 | 13 | 0 | 0 |
| 52 | ----- 60 | 16 | 16 | 3 | 3 |
| 52 | ----- 61 | 13 | 0 | 0 | 0 |
| 53 | ----- 61 | 27 | 14 | 0 | 0 |
| 53 | ----- 62 | 54 | 0 | 0 | 27 |
| 54 | ----- 63 | 45 | 13 | 0 | 13 |
| 54 | ----- 55 | 32 | 0 | 0 | 0 |
| 55 | ----- 63 | 32 | 0 | 0 | 0 |
| 56 | ----- 63 | 44 | 12 | 0 | 15 |
| 56 | ----- 73 | 62 | 17 | 0 | 33 |
| 56 | ----- 64 | 63 | 0 | 0 | 34 |
| 56 | ----- 65 | 29 | 0 | 0 | 0 |
| 57 | ----- 65 | 42 | 13 | 0 | 0 |
| 58 | ----- 66 | 24 | 0 | 0 | 0 |
| 58 | ----- 67 | 31 | 0 | 0 | 7 |
| 59 | ----- 67 | 36 | 5 | 0 | 17 |

| | | | | |
|---|---|---|---|---|
| 59 ----- 68 | 19 | 19 | 0 | 0 |
| 60 ----- 68 | 10 | 10 | 10 | 10 |
| 60 ----- 69 | 0 | 0 | 0 | 0 |
| 61 ----- 69 | 13 | 13 | 0 | 0 |
| 61 ----- 70 | 49 | 0 | 0 | 36 |
| 62 ----- 70 | 54 | 5 | 0 | 0 |
| 62 ----- 71 | 68 | 0 | 0 | 14 |
| 62 ----- 72 | 70 | 38 | 0 | 16 |
| 63 ----- 72 | 32 | 0 | 0 | 0 |
| 63 ----- 73 | 45 | 0 | 0 | 13 |
| 64 ----- 73 | 63 | 18 | 0 | 0 |
| 65 ----- 74 | 29 | 0 | 0 | 0 |
| 65 ----- 83 | 39 | 10 | 0 | 10 |
| 66 ----- 75 | 24 | 0 | 0 | 0 |
| 66 ----- 76 | 42 | 11 | 0 | 18 |
| 67 ----- 76 | 31 | 0 | 0 | 0 |
| 68 ----- 77 | 16 | 0 | 0 | 16 |
| 68 ----- 78 | 0 | 0 | 0 | 0 |
| 69 ----- 68 | 0 | 0 | 0 | 0 |
| 69 ----- 78 | 29 | 29 | 29 | 29 |
| 69 ----- 79 | 38 | 0 | 0 | 38 |
| 70 ----- 80 | 49 | 13 | 0 | 0 |
| 71 ----- 80 | 68 | 32 | 0 | 0 |
| 72 ----- 81 | 32 | 0 | 0 | 0 |
| 73 ----- 81 | 45 | 13 | 0 | 0 |
| 73 ----- 82 | 63 | 30 | 0 | 18 |
| 74 ----- 82 | 57 | 24 | 0 | 28 |
| 74 ----- 83 | 29 | 0 | 0 | 0 |
| 75 ----- 84 | 24 | 0 | 0 | 0 |
| 76 ----- 84 | 31 | 7 | 0 | 0 |
| 76 ----- 85 | 32 | 10 | 0 | 1 |
| 77 ----- 85 | 22 | 0 | 0 | 6 |
| 78 ----- 86 | 8 | 0 | 0 | 8 |
| 78 ----- 87 | 0 | 0 | 0 | 0 |
| 79 ----- 87 | 38 | 38 | 0 | 0 |
| 79 ----- 88 | 44 | 8 | 0 | 6 |
| 80 ----- 88 | 36 | 0 | 0 | 0 |
| 80 -----100 | 48 | 48 | 12 | 12 |
| 80 ----- 89 | 45 | 0 | 0 | 9 |
| 81 ----- 89 | 49 | 4 | 0 | 17 |
| 81 ----- 90 | 35 | 0 | 0 | 3 |
| 81 ----- 91 | 32 | 0 | 0 | 0 |
| 82 ----- 92 | 33 | 0 | 0 | 0 |
| 83 ----- 82 | 33 | 0 | 0 | 4 |
| 83 ----- 92 | 52 | 19 | 0 | 23 |
| 83 ----- 93 | 29 | 0 | 0 | 0 |
| 84 ----- 94 | 31 | 8 | 0 | 7 |
| 85 ----- 94 | 23 | 0 | 0 | 1 |
| 85 ----- 95 | 22 | 0 | 0 | 0 |
| 85 ----- 96 | 27 | 0 | 0 | 5 |
| 86 ----- 96 | 28 | 1 | 0 | 20 |
| 86 ----- 97 | 8 | 0 | 0 | 0 |
| 87 ----- 97 | 8 | 0 | 0 | 8 |
| 87 ----- 98 | 7 | 0 | 0 | 7 |
| 87 ----- 99 | 0 | 0 | 0 | 0 |
| 88 ----- 99 | 36 | 36 | 0 | 0 |
| 89 -----100 | 45 | 45 | 0 | 0 |
| 89 -----101 | 45 | 13 | 0 | 0 |
| 90 -----101 | 35 | 3 | 0 | 0 |
| 91 -----101 | 32 | 0 | 0 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| 93 | -----103 | 29 | 0 | 0 | 0 |
| 93 | -----104 | 54 | 0 | 0 | 25 |
| 94 | -----105 | 23 | 1 | 0 | 0 |
| 95 | -----105 | 22 | 0 | 0 | 0 |
| 96 | -----105 | 27 | 5 | 0 | 0 |
| 96 | -----106 | 29 | 0 | 0 | 2 |
| 97 | -----107 | 15 | 0 | 0 | 7 |
| 97 | -----108 | 8 | 1 | 0 | 0 |
| 98 | -----108 | 7 | 0 | 0 | 0 |
| 98 | -----109 | 9 | 3 | 0 | 2 |
| 99 | -----109 | 6 | 0 | 0 | 6 |
| 99 | -----110 | 4 | 0 | 0 | 4 |
| 99 | -----100 | 0 | 0 | 0 | 0 |
| 100 | -----110 | 5 | 1 | 1 | 5 |
| 100 | -----111 | 2 | 0 | 0 | 2 |
| 100 | -----112 | 0 | 0 | 0 | 0 |
| 101 | -----112 | 32 | 32 | 0 | 0 |
| 101 | -----113 | 40 | 7 | 0 | 8 |
| 102 | -----113 | 33 | 0 | 0 | 0 |
| 103 | -----113 | 44 | 11 | 0 | 15 |
| 103 | -----114 | 29 | 0 | 0 | 0 |
| 104 | -----112 | 54 | 54 | 0 | 0 |
| 105 | -----115 | 22 | 0 | 0 | 0 |
| 105 | -----116 | 44 | 0 | 0 | 22 |
| 106 | -----116 | 44 | 0 | 0 | 15 |
| 106 | -----117 | 29 | 0 | 0 | 0 |
| 106 | -----118 | 31 | 16 | 0 | 2 |
| 107 | -----118 | 15 | 0 | 0 | 0 |
| 108 | -----118 | 34 | 19 | 12 | 27 |
| 108 | -----119 | 7 | 0 | 0 | 0 |
| 109 | -----119 | 8 | 1 | 0 | 2 |
| 109 | -----120 | 6 | 2 | 0 | 0 |
| 110 | -----120 | 4 | 0 | 0 | 0 |
| 110 | -----121 | 9 | 0 | 0 | 5 |
| 111 | -----121 | 13 | 4 | 2 | 11 |
| 111 | -----130 | 31 | 31 | 29 | 29 |
| 111 | -----122 | 2 | 2 | 0 | 0 |
| 112 | -----122 | 0 | 0 | 0 | 0 |
| 112 | -----123 | 7 | 0 | 0 | 7 |
| 113 | -----123 | 33 | 26 | 0 | 0 |
| 114 | -----124 | 29 | 0 | 0 | 0 |
| 115 | -----135 | 34 | 4 | 0 | 12 |
| 115 | -----125 | 22 | 0 | 0 | 0 |
| 116 | -----125 | 44 | 22 | 0 | 0 |
| 116 | -----126 | 46 | 4 | 0 | 2 |
| 117 | -----126 | 42 | 0 | 0 | 13 |
| 117 | -----127 | 29 | 14 | 0 | 0 |
| 118 | -----127 | 15 | 0 | 0 | 0 |
| 119 | -----127 | 25 | 10 | 3 | 18 |
| 119 | -----139 | 32 | 9 | 2 | 25 |
| 119 | -----128 | 7 | 3 | 0 | 0 |
| 120 | -----128 | 4 | 0 | 0 | 0 |
| 120 | -----129 | 7 | 0 | 0 | 3 |
| 121 | -----129 | 9 | 2 | 0 | 0 |
| 121 | -----130 | 18 | 18 | 9 | 9 |
| 122 | -----130 | 0 | 0 | 0 | 0 |
| 122 | -----131 | 12 | 0 | 0 | 12 |
| 122 | -----132 | 25 | 0 | 0 | 25 |
| 123 | -----132 | 30 | 5 | 0 | 23 |

| | | | | | |
|---|---|---|---|---|---|
| 123 | -----133 | 7 | 0 | 0 | 0 |
| 124 | -----123 | 29 | 22 | 0 | 0 |
| 124 | -----133 | 46 | 39 | 10 | 17 |
| 124 | -----134 | 77 | 0 | 0 | 48 |
| 125 | -----135 | 30 | 0 | 0 | 8 |
| 125 | -----136 | 22 | 0 | 0 | 0 |
| 126 | -----137 | 42 | 20 | 0 | 0 |
| 127 | -----137 | 22 | 0 | 0 | 7 |
| 127 | -----138 | 15 | 0 | 0 | 0 |
| 127 | -----139 | 23 | 0 | 0 | 8 |
| 128 | -----139 | 23 | 0 | 0 | 19 |
| 128 | -----140 | 28 | 0 | 0 | 24 |
| 128 | -----141 | 4 | 0 | 0 | 0 |
| 129 | -----141 | 7 | 3 | 0 | 0 |
| 129 | -----142 | 25 | 25 | 18 | 18 |
| 130 | -----142 | 0 | 0 | 0 | 0 |
| 131 | -----142 | 12 | 12 | 0 | 0 |
| 131 | -----143 | 15 | 0 | 0 | 3 |
| 132 | -----143 | 25 | 10 | 0 | 0 |
| 133 | -----144 | 7 | 0 | 0 | 0 |
| 133 | -----145 | 26 | 0 | 0 | 19 |
| 134 | -----146 | 77 | 39 | 0 | 0 |
| 135 | -----147 | 30 | 8 | 0 | 0 |
| 136 | -----147 | 22 | 0 | 0 | 0 |
| 136 | -----148 | 28 | 6 | 0 | 6 |
| 137 | -----148 | 22 | 0 | 0 | 0 |
| 137 | -----149 | 24 | 9 | 0 | 2 |
| 138 | -----149 | 15 | 0 | 0 | 0 |
| 139 | -----149 | 23 | 8 | 0 | 0 |
| 139 | -----150 | 38 | 5 | 0 | 15 |
| 140 | -----150 | 33 | 0 | 0 | 5 |
| 140 | -----151 | 28 | 0 | 0 | 0 |
| 141 | -----152 | 4 | 4 | 0 | 0 |
| 142 | -----152 | 0 | 0 | 0 | 0 |
| 143 | -----152 | 15 | 15 | 0 | 0 |
| 143 | -----153 | 20 | 13 | 0 | 5 |
| 144 | -----153 | 7 | 0 | 0 | 0 |
| 144 | -----154 | 24 | 0 | 0 | 17 |
| 145 | -----154 | 26 | 2 | 0 | 0 |
| 146 | -----154 | 38 | 14 | 0 | 0 |
| 147 | -----155 | 22 | 7 | 0 | 0 |
| 148 | -----155 | 22 | 7 | 0 | 0 |
| 149 | -----155 | 15 | 0 | 0 | 0 |
| 150 | -----157 | 33 | 18 | 0 | 0 |
| 151 | -----157 | 28 | 13 | 0 | 0 |
| 152 | -----158 | 0 | 0 | 0 | 0 |
| 153 | -----159 | 7 | 0 | 0 | 0 |
| 154 | -----159 | 24 | 17 | 0 | 0 |
| 155 | -----156 | 15 | 0 | 0 | 0 |
| 156 | -----157 | 15 | 0 | 0 | 0 |
| 157 | -----160 | 15 | 15 | 0 | 0 |
| 158 | -----160 | 0 | 0 | 0 | 0 |
| 159 | -----160 | 7 | 7 | 0 | 0 |

REFERENCES

1.Malcolm,D.G.,Fazar,W.,et al,(1959) "Application of a Technique for R&D Program Evaluation",Systems Dev. Corp.,Santa Monica CA

2.Kelley,J.E.,and Walker,M.R.,(1959) "Critical Path Planning and Scheduling",Proc. of the Eastern Joint Computer Conference, pp. 160-173

3.Siemens,N.,(1971) "A Simple CPM Time-Cost Trade off Algorithm" Management Science,Vol. 17,pp. 354-363

4.Fulkerson,D.R.,(1961) "A Network Flow Computation for Project Cost Curves",Management Science,Vol. 7,pp. 167-168

5.Golomb,S.W.,and Baumert,L.D.,(1965) "Backtrack Programming", J.ACM,Vol. 12,pp. 516-524

6.Tarjan,R.,(1972) "Depth-first Search and Linear Graph Algorithms",SIAM J.COMPUT.,Vol. 1(2),pp. 146-160

7.Berztiss,A.T.,(1980) "Depth-first K-trees and Critical Path Analysis",Acta Informatica,Vol. 13,pp. 325-346

8.Satyanarayana,A.,(1982) "A Unified Formula for Analysis of Some Network Reliability Problems" ,IEEE Transactions on Reliability,Vol. R-31,pp. 23-32

9.Locks,M.O.,(1983) "Optimal Search for Critical Path",unpub.

10.Smith,L.A.,and Mahler,P.,(1978) "Comparing Commercially Available CPM/PERT Computer Programs",Industrial Engineering, Vol. 10,pp. 37-39

11.Thomas,W.,(1969) "Four Float Measures for Critical Path Scheduling",Industrial Engineering,Vol. 1,pp. 19-23

12.Phillips,D.T.,and Garcia-Diaz,A.,(1981) "Fundamentals of Network Analysis",Prenctice Hall Inc.

13.PREMIS - Commercially available software package for CPM

14.Ashbrook,M.,and Zinn,H.,(1980) "A First Look at Graph Theory Applications",BYTE,Vol. 5(2),pp. 18-28

# APPENDIX

```
/*   THIS PROGRAM EXECUTES A SINGLE PASS,DEPTH-FIRST SEARCH METHOD      *
/*   FOR FINDING THE CRITICAL PATH IN A DIRECTED,ACYCLIC NETWORK.       *
/*   THE PROGRAM CAN BE USED TO SOLVE PROBLEMS RELATING TO A SIMPLE     *
/*   NETWORK OR TO A SUPERNETWORK OF SEVERAL MODULES,EACH MODULE        *
/*   BEING A SIMPLE NETWORK IN ITSELF WITH ONE STARTING AND ONE         *
/*   ENDING EVENT.                                                      *
/*   THE PROGRAM CAN HANDLE UPTO 50 MODULES AT A TIME.                  *
/*   ALSO,IT FINDS MULTIPLE CRITICAL PATHS IF MORE THAN ONE EXIST.      *
/*   HOWEVER,THE NUMBER OF PARTIAL CRITICAL PATHS FROM ANY ONE EVENT    *
/*   CANNOT EXCEED 3.IN PRACTICAL PROBLEMS THIS LIMITATION CANNOT BE    *
/*   OF ANY CONSEQUENCE.                                                *
/*   THE PROGRAM CONSISTS OF A MAIN PROCEDURE,DFSCRIP,AND SIX           *
/*   INTERNAL PROCEDURES.A DESCRIPTION OF THE FUNCTIONS OF EACH         *
/*   PROCEDURE IS GIVEN AT ITS BEGINNING.                               *
/*   THOUGH THE DEPTH FIRST SEARCH REQUIRES ONLY ONE PATH TO FIND THE   *
/*   CRITICAL PATH,TWO PASSES ARE REQUIRED TO BE ABLE TO CALCULATE      *
/*   THE FLOAT VALUES.THIS PROGRAM HAS BEEN WRITTEN TO INCLUDE THE      *
/*   FLOAT CALCULATIONS.                                                *
/*------------INSTRUCTIONS  FOR  ENTERING  DATA--------------------*
/*                                                                      *
/*   THE INPUT DATA IS TO BE ENTERED AT THE END OF THE PROGRAM AS       *
/*   DESCRIBED HERE.                                                    *
/*   DATA FOR EACH OF THE MODULES IS ENTERED FIRST AND THEN THE DATA    *
/*   FOR THE SUPERNETWORK.                                              *
/*   FOR EACH MODULE:                                                   *
/*   ENTER THE MODULE NAME WITHIN SINGLE QUOTES FOLLOWED BY THE         *
/*   NUMBER OF EVENTS.THE MODULE NAME CANNOT EXCEED 15 CHARACTERS       *
/*   INCLUDING BLANKS.THEN, FOR EACH EVENT(EXCEPT THE FINISHING         *
/*   EVENT) ENTER EACH SUCCESSOR EVENT AND CORRESPONDING ACTIVITY       *
/*   TIME.ENTER A 0 AT THE END OF THE LIST FOR EACH EVENT.EXAMPLE--     *
/*     'JOB 1'  ,  5                                                    *
/*   1 , 2 , A , 4 , B , 3 , C , 0                                      *
/*   3 , 2 , E , 4 , 0 , 0                                              *
/*   2 , 5 , D , 0                                                      *
/*   4 , 5 , G , 0                                                      *
/*   THE FIRST LINE STATES THAT  JOB 1 HAS 5 EVENTS.THE SECOND LINE     *
/*   STATES THAT EVENT 1 IS SUCCEEDED BY EVENT 2,4,AND 3 AND THE        *
/*   THE CORRESPONDING ACTIVITY TIMES ARE A,B AND C RESPECTIVELY.       *
/*   LIKEWISE,FROM THE SUBSEQUENT LINES IT IS INFERRED THAT ACTIVITY    *
/*   2-5 HAS A TIME OF D,ACTIVITY 3--2  HAS A TIME OF 4 AND SO ON.      *
/*   SINCE THERE ARE NO ACTIVITIES STARTING FROM EVENT 5,THERE WILL     *
/*   BE NO CORRESPONDING LINE OF ENTRIES.                               *
/*   DUMMY ACTIVITIES ARE ENTERED LIKE ALL OTHER ACTIVITES,WITH AN      *
/*   ACTIVITY TIME OF 0.ACTIVITY 3--4 IN THE EXAMPLE ABOVE IS A DUMMY   *
/*   ACTIVITY.                                                          *
/*   THE ORDER OF ENTRY OF EVENTS WITHIN A LINE OR AMONG LINES IS       *
/*   INCONSEQUENTIAL.IN THE EXAMPLE ABOVE,EVENT 4 APPEARS BEFORE        *
/*   EVENT 3 IN LINE 2.ALSO, ENTRIES FOR STARTING EVENT 3 HAS BEEN      *
/*   MADE BEFORE ENTRIES FOR EVENT 2.                                   *
/*   FOR INTEGRATING THE MODULES ALSO A SIMILAR PATTERN OF DATA         *
/*   ENTRY IS USED.HOWEVER,THE NAME MUST BE ENTERED AS 'PROJECT' AND    *
/*   NO OTHER NAME MAY BE USED.FURTHER,THE ACTIVITY TIMES ARE TO BE     *
/*   REPLACED BY MODULE NAMES.EXAMPLE--                                 *
/*   'PROJECT'  ,  6                                                    *
```

```
/*  1 , 2 , 'JOB 1' , 4 , 'JOB 3' , 0                                    *
/*  AND SO ON FOR THE REMAINING EVENTS.                                  *
/*  THIS MEANS THAT THERE ARE 6 EVENTS IN THE SUPERNETWORK.EVENTS        *
/*  1 AND 2 ARE CONNECTED BY JOB 1,EVENTS 1 AND 4 BY JOB 3 ETC.          *
/*  AS IN THE CASE OF MODULES THERE WILL BE ENTRIES CORRESPONDING        *
/*  TO 5 EVENTS,ONE LESS THAN THE TOTAL NUMBER OF EVENTS.                *
/*  THE NAME OF THE MODULES HERE AND THE INDIVIDUAL MODULE DATA          *
/*  SHOULD BE IDENTICAL WITHIN QUOTES,INCLUDING BLANK SPACES.FOR         *
/*  EXAMPLE, ' JOB 1' WILL NOT BE IDENTIFIED AS 'JOB 1'.                 *
/*  ALL DATA CAN BE ENTERED IN FREE FORMAT.THERE ARE NO SPECIFIED        *
/*  COLOUMNS OR FIELDS FOR ENTRY.NUMBERS MUST BE SEPARATED BY            *
/*  ATLEAST ONE BLANK OR A COMMA.NAMES MUST APPEAR WITHIN QUOTES.        *
/*  ZEROS AT THE END OF THE LISTS OF ACTIVITY TIMES ARE NECESSARY.       *
/*                                                                       *
/*-----------------------------------------------------------------------*
/*                                                                       *
/*-----------------------------------------------------------------------*
DFSCRIP:PROCEDURE OPTIONS(MAIN);

/*  THIS MAIN PROCEDURE READS THE NAMES OF THE MODULES AND THE           *
/*  THE NUMBER  OF EVENTS IN THE MODULES AND BY INVOKING THE PROCE-      *
/*  DURE 'SEARCH',FINDS THE SOLUTION TO THE CRITICAL PATH PROBLEM.       *


DCL TOTAL_EVENTS FIXED;
DCL 1 MODULE(50),
      2 NAME CHAR(15) VAR,
      2 TIME FIXED;
DCL(A,B,C,I,J,K,M,P,X,Y,Z) FIXED;

M=1;
DO WHILE('1'B);
   GET LIST (MODULE(M).NAME,TOTAL_EVENTS);
   CALL SEARCH;
   IF NAME(M)='PROJECT' THEN RETURN;
   M=M+1;
END;

RETURN;
/*-----------------------------------------------------------------------*

SEARCH:PROCEDURE;

/*  THIS PROCEDURE CARRIES OUT THE DFS SEARCH.THE FIRST SECTION OF       *
/*  THIS PROCEDURE CONSTRUCTS THE FORWARD AND BACKWARD LINKED LISTS      *
/*  FOR STORING THE INFORMATION ABOUT THE NETWORK.THE SECOND SEC-        *
/*  TION BUILDS THE TREE BY CALLING THE 'BUILD_TREE' PROCEDURE,DE-       *
/*  TERMINES THE EVENT TIMES(EET AND LET) BY CALLING THE PROCEDURE       *
/*  'CRITICAL_PATH',AND FINDS THE CRITICAL PATH BY TRACING THE           *
/*  CRITICAL EVENTS BY MEANS OF THE 'CP_ROUTE' PROCEDURE.THE THIRD       *
/*  SECTION CALCULATES THE VARIOUS FLOAT VALUES.                         *
/*  THE DEFINITIONS OF MOST OF THE VARIABLES USED IN THIS PROGRAM        *
/*  ARE SELF EVIDENT.THE STRUCTURED VARIABLES ARE DESCRIBED BRIEFLY      *
/*  HERE.                                                                *
```

```
/*  TREENODE - REPRESENTS THE NODE ON THE TREE.                          *
/*             .EVENT STANDS FOR THE CORRESPONDING EVENT ON THE NET-     *
/*                   WORK;                                               *
/*             .PATH  STANDS FOR THE LENGTH OF THE PARTIAL PATH FROM     *
/*                   THAT EVENT TO THE TERMINAL;                         *
/*             .SON AND .BROTHER ARE USED TO IDENTIFY THE RELATIONS      *
/*                   BETWEEN THE NODES ON THE TREE.                      *
/*  HEADER(X) - FLINK AND BLINK POINT TO THE FIRST  CARDS ON THE         *
/*              FORWARD AND BACKWARD LINKED LISTS FOR THE EVENT X.       *
/*  CARD      - USED TO STORE INFORMATION ABOUT THE INTERRELATION-       *
/*              SHIPS BETWEEN THE EVENTS ON THE NETWORK.                 *
/*              .EVENT STANDS FOR A SUCCESSOR OR PREDECESSOR EVENT Y     *
/*                    OF EVENT X.                                        *
/*              .TIME STANDS FOR THE DURATION OF ACTIVITY X--Y.          *
/*              .LINK POINTS TO THE NEXT CARD ON THE LINKED LIST.        *
/*                                                                       *
/*-----------------------------------------------------------------------*

DCL 1 TREENODE(3*TOTAL_EVENTS),
      ( 2 EVENT,
        2 PATH,
        2 FATHER,
        2 SON,
        2 BROTHER) FIXED;

DCL 1 HEADER(TOTAL_EVENTS),
      ( 2 FLINK,
        2 BLINK ) FIXED;

DCL 1 CARD(5*TOTAL_EVENTS),
      (2 EVENT,
       2 TIME,
       2 LINK ) FIXED;

DCL (EET(TOTAL_EVENTS),
     LET(TOTAL_EVENTS),
     SUCCESSOR(TOTAL_EVENTS,3),
     FLAG(TOTAL_EVENTS) )        FIXED;

DCL SOURCE FIXED;

DCL (FREE_FLOAT,
     INDEP_FLOAT,
     SAFETY_FLOAT,
     TOTAL_FLOAT) FIXED;


DCL DIRECTION CHAR(10) VAR,
    MODULE CHAR(15) VAR;


DO I=1 TO TOTAL_EVENTS;
   EET(I),LET(I)=0;
   HEADER(I).FLINK=0;
```

```
   HEADER(I).BLINK=0;
   DO J=1 TO 3;
      SUCCESSOR(I,J)=0;
   END;
END;

/*                                                SECTION 1 -----------*
A=0;
DO J=1 TO TOTAL_EVENTS-1;
   GET LIST(X);
   IF (NAME(M)='PROJECT' & M>1) THEN GET LIST(Y,MODULE);
   ELSE GET LIST(Y,Z);
   A=A+1;
   LOOP:;
   DIRECTION='FORWARD';
   CALL TRACE(X,DIRECTION);
   IF B=0 THEN HEADER(X).FLINK=A;
      ELSE CARD(B).LINK=A;
   CARD(A).EVENT=Y;
   CARD(A).LINK=0;
   IF (NAME(M)='PROJECT' & M>1) THEN CALL IDENTIFY(MODULE,Z);
   CARD(A).TIME=Z;
   A=A+1;
   DIRECTION='BACKWARD';
   CALL TRACE(Y,DIRECTION);
   IF(B=0) THEN HEADER(Y).BLINK=A;
     ELSE CARD(B).LINK=A;
   CARD(A).EVENT=X;
   CARD(A).LINK=0;
   CARD(A).TIME=Z;
   JUMP:;
   GET LIST(Y);
       IF (Y=0) THEN GOTO OUT;
   IF (NAME(M)='PROJECT' & M>1) THEN GET LIST(MODULE);
   ELSE GET LIST(Z);
   A=A+1;
   GOTO LOOP;
   OUT:;
END;
/*                                                ----------------------*

/*                                                SECTION 2 ------------*
DO L=2 TO 1 BY -1;
   DO J=1 TO TOTAL_EVENTS;
      FLAG(J)=0;
   END;
   IF L=1 THEN SOURCE=1;
      ELSE SOURCE=TOTAL_EVENTS;
   C=1;
   TREENODE(C).EVENT=SOURCE;
   TREENODE(C).PATH=0;
   TREENODE(C).SON=0;
```

```
   TREENODE(C).BROTHER=0;
   IF L=1 THEN DO;
                  CALL BUILD_TREE(SOURCE,C);
                  I=1;
                  CALL CRITICALPATH(I,LET,FLINK);
               END;
      ELSE DO;
              CALL BUILD_TREE(SOURCE,C);
              I=1;
              CALL CRITICALPATH(I,EET,BLINK);
           END;
END;

PUT PAGE EDIT(MODULE(M).NAME)(X(30),A);

PUT SKIP(3)EDIT('CRITICAL PATH LENGTH =',LET(1))
              (COL(5),A,COL(40),F(4));
MODULE(M).TIME=LET(1);



PUT SKIP(3)EDIT('CRITICAL PATH  ')(COL(5),A);
K=1;
Y=0;
CALL CP_ROUTE(K);
/*                                                    -------------------


PUT SKIP(5)EDIT('EVENT','EVENT FLOAT')(COL(20),A,COL(35),A);
DO K=1 TO TOTAL_EVENTS;
   PUT SKIP EDIT(K,LET(1)-LET(K)-EET(K))
                (COL(21),F(3),COL(38),F(3));
END;
PUT PAGE   EDIT('ACTIVITY','TOTAL FLOAT','FREE FLOAT','INDEP. FLOAT',
                'SAFETY FLOAT')
               (COL(10),A,COL(25),A,COL(40),A,COL(55),A,COL(70),A);

DO I=1 TO TOTAL_EVENTS;
   K=HEADER(I).FLINK;
   DO WHILE(K¬=0);
      IF CARD(K).TIME=0 THEN GOTO NEXT;
      J=CARD(K).EVENT;
      TOTAL_FLOAT=LET(1)-LET(J)-EET(I)-CARD(K).TIME;
      FREE_FLOAT=EET(J)-EET(I)-CARD(K).TIME;
      INDEP_FLOAT=MAX(0,EET(J)-LET(1)+LET(I)-
                        CARD(K).TIME);
      SAFETY_FLOAT=LET(I)-LET(J)-CARD(K).TIME;
      PUT SKIP EDIT(I,'-----',J,TOTAL_FLOAT,FREE_FLOAT,INDEP_FLOAT,
                     SAFETY_FLOAT)
                    (COL(7),F(3),COL(12),A,COL(17),F(3),COL(28),F(3),
                     COL(43),F(3),COL(57),F(3),COL(72),F(3));

      NEXT:;
```

```
      K=CARD(K).LINK;
   END;
END;
/*                                                          ---------------*
RETURN;

/*---------------------------------------------------------------------------*
TRACE:PROCEDURE(EVENT,DIRECTION);

/*  THIS PROCEDURE FINDS THE LAST CARD ON THE LINKED LIST FOR THE          *
/*  EVENT SPECIFIED WITHIN PARANTHESIS.DIRECTION SPECIFIES THE             *
/*  FORWARD OR THE BACKWARD LIST.                                          *

DCL (EVENT,A)FIXED;
DCL DIRECTION CHAR(*) VAR;

IF (DIRECTION='FORWARD') THEN A=HEADER(EVENT).FLINK;
   ELSE A=HEADER(EVENT).BLINK;
IF(A=0) THEN B=0;
  ELSE DO WHILE(A¬=0);
          B=A;
          A=CARD(A).LINK;
       END;

RETURN;
END TRACE;
/*---------------------------------------------------------------------------*

BUILD_TREE:PROCEDURE(EVENT,TREE_NODE)RECURSIVE;

/*  FOR AN EVENT CORRESPONDING TO THE SPECIFIED TREE-NODE,THIS             *
/*  PROCEDURE IDENTIFIES THE CORRESPONDING SUCCESSOR EVENTS ON THE  *,
/*  NETWORK AND ADDS BRANCHES ON THE TREE.                                 *

DCL (EVENT,TREE_NODE,DUMMY)FIXED;
DCL (D,E,F) FIXED;

D=TREE_NODE;
IF FLAG(EVENT)=1 THEN RETURN;
IF( (EVENT=1 & L=2)|(EVENT=TOTAL_EVENTS & L=1) )
   THEN DO;
          FLAG(EVENT)=1;
          RETURN;
       END;
IF L=1 THEN F=HEADER(EVENT).FLINK;
   ELSE F=HEADER(EVENT).BLINK;
   C=C+1;
   TREENODE(C).EVENT=CARD(F).EVENT;
   TREENODE(C).BROTHER=0;
   TREENODE(C).SON=0;
   TREENODE(D).SON=C;
F=CARD(F).LINK;
```

```
DO WHILE(F¬=0);
   C=C+1;
   TREENODE(C).EVENT=CARD(F).EVENT;
   TREENODE(C).SON=0;
   TREENODE(C).BROTHER=0;
   TREENODE(C-1).BROTHER=C;
   TREENODE(C).FATHER=D;
   F=CARD(F).LINK;
END;

F=TREENODE(D).SON;
DO WHILE(F¬=0);
   CALL BUILD_TREE(TREENODE(F).EVENT,F);
   F=TREENODE(F).BROTHER;
END;
   FLAG(EVENT)=1;

RETURN;
END BUILD_TREE;
/*-----------------------------------------------------------------------

CRITICALPATH:PROCEDURE(NODE,EVENT_TIME,LINK)RECURSIVE;

/*  STARTING FROM A SPECIFIED NODE ON THE TREE,THIS PROCEDURE CAL-       :
/*  CULATES THE LONGEST PARTIAL PATH BETWEEN THE CORRESPONDING EVENT     :
/*  AND THE TERMINAL.ALSO IT DETERMINES THE CORRECT SEQUENCE OF          :
/*  ACTIVITIES ALONG THIS PATH.                                          :
/*  THE ARGUMENT LINK IDENTIFIES THE DIRECTION OF THE PASS AND  THE      :
/*  APPROPRIATE EVENT TIME IS OUTPUT AS THE SECOND ARGUMENT.             :

DCL (A,B,C,DUMMY) FIXED;
DCL (NODE,
     EVENT_TIME(*),
     LINK(*) ) FIXED;

DUMMY=TREENODE(NODE).EVENT;
IF (DUMMY=TOTAL_EVENTS & L=1) | (DUMMY=1 & L=2)
   THEN DO;
        EVENT_TIME(DUMMY)=0;
        RETURN;
     END;

A=TREENODE(NODE).SON;
DO WHILE(A¬=0);
   CALL CRITICALPATH(A,EVENT_TIME,LINK);
   A=TREENODE(A).BROTHER;
END;

A=TREENODE(NODE).SON;
IF A=0 THEN RETURN;
B=LINK(DUMMY);
C=0;
DO WHILE(A¬=0);
```

```
   IF EVENT_TIME(DUMMY)< EVENT_TIME(TREENODE(A).EVENT)+CARD(B).TIME
      THEN DO;
             EVENT_TIME(DUMMY)=
             EVENT_TIME(TREENODE(A).EVENT)+CARD(B).TIME;
             C=1;
             SUCCESSOR(DUMMY,C)=TREENODE(A).EVENT;
          END;
   ELSE IF EVENT_TIME(DUMMY)=EVENT_TIME(TREENODE(A).EVENT)+CARD(B).TIM
      THEN DO;
              C=C+1;
              SUCCESSOR(DUMMY,C)=TREENODE(A).EVENT;
           END;
   A=TREENODE(A).BROTHER;
   B=CARD(B).LINK;
END;

DO A=C+1 TO 3;
   SUCCESSOR(DUMMY,A)=0;
END;

RETURN;
END CRITICALPATH;
/*-------------------------------------------------------------------

CP_ROUTE:PROCEDURE(EVENT) RECURSIVE;

/*  STARTING FROM THE EVENT SPECIFIED AS THE ARGUMENT,THIS PROCEDURE
/*  TRACES THE PARTIAL PATH FROM THAT EVENT TO THE TERMINAL.
/*  IF THE EVENT SPECIFIED IS THE STARTING EVENT,THEN THE
/*  CRITICAL PATH IS OBTAINED.

DCL EVENT FIXED;
DCL X FIXED;

IF(EVENT=TOTAL_EVENTS) THEN DO;
                              Y=Y+1;
                              IF Y=8 THEN Y=1;
                              PUT EDIT(EVENT)(COL(20+Y*5),F(3));
                              RETURN;
                           END;
X=1;
DO WHILE(SUCCESSOR(EVENT,X)¬=0);
   Y=Y+1;
   IF Y=8 THEN Y=1;
   PUT EDIT (EVENT)(COL(20+5*Y),F(3));
   CALL CP_ROUTE(SUCCESSOR(EVENT,X));
   X=X+1;
END;

RETURN;
END CP_ROUTE;
/*-------------------------------------------------------------------
IDENTIFY:PROCEDURE(MODULE_NAME,MODULE_TIME);
```

```
/*   THIS PROCEDURE RETRIEVES THE CALCULATED CRITICAL PATH FOR A        *
/*   MODULE SPECIFIED BY ITS NAME.THOUGH THE USER CAN IDENTIFY          *
/*   MODULES BY ALPHA-NUMERIC CHARACTERS,THE PROGRAM ASSIGNS NUMBERS    *
/*   TO THE MODULES AND WHEN A MODULE IS REFERRED TO BY ITS NAME,       *
/*   A SUITABLE IDENTIFICATION PROCEDURE IS REQUIRED.                   *


DCL MODULE_NAME CHAR(*) VAR,
    MODULE_TIME FIXED;

DO L=1 TO M-1;
   IF NAME(L)=MODULE_NAME
      THEN DO;
              MODULE_TIME=MODULE(L).TIME;
              RETURN;
           END;
END;

RETURN;
END IDENTIFY;
/*----------------------------------------------------------------------*
END SEARCH;
/*----------------------------------------------------------------------*


END DFSCRIP;
/*----------------------------------------------------------------------*

/*----------------------------------------------------------------------*
/* THE INPUT DATA IS TO BE ENTERED NOW AFTER THE  '*DATA' CARD.
   SEE INSTRUCTIONS AT THE BEGINNING OF THE PROGRAM.                    *

*DATA
```

```
/*   THIS PROGRAM EXECUTES THE CONVENTIONAL,TWO PASS METHOD FOR          *
/*   FINDING THE CRITICAL PATH IN A DIRECTED,ACYCLIC NETWORK.            *
/*   THE PROGRAM CAN BE USED TO SOLVE PROBLEMS RELATING TO A SINGLE      *
/*   ACTIVITY NETWORK OR TO A SUPER NETWORK OF SEVERAL MODULES,EACH      *
/*   MODULE BEING A SIMPLE NETWORK WITH ONE STARTING EVENT AND ONE       *
/*   ENDING EVENT.                                                       *
/*   THE PROGRAM CAN HANDLE UPTO 50 MODULES AT A TIME.                   *
/*   THE PROGRAM CONSISTS OF A MAIN PROCEDURE,CONCRIP,AND SIX            *
/*   INTERNAL PROCEDURES.A DESCRIPTION OF THE FUNCTIONS OF EACH          *
/*   PROCEDURE IS GIVEN AT ITS BEGINNING.                                *

/*------------INSTRUCTIONS  FOR  ENTERING  DATA--------------------*
/*                                                                       *
/*   THE INPUT DATA IS TO BE ENTERED AT THE END OF THE PROGRAM AS        *
/*   DESCRIBED HERE.                                                     *
/*   DATA FOR EACH OF THE MODULES IS ENTERED FIRST AND THEN THE DATA     *
/*   FOR THE SUPERNETWORK.                                               *
/*   FOR EACH MODULE:                                                    *
/*   ENTER THE MODULE NAME WITHIN SINGLE QUOTES FOLLOWED BY THE          *
/*   NUMBER OF EVENTS.THE MODULE NAME CANNOT EXCEED 15 CHARACTERS        *
/*   INCLUDING BLANKS.THEN, FOR EACH EVENT(EXCEPT THE FINISHING          *
/*   EVENT) ENTER EACH SUCCESSOR EVENT AND CORRESPONDING ACTIVITY        *
/*   TIME.ENTER A 0 AT THE END OF THE LIST FOR EACH EVENT.EXAMPLE--      *
/*     'JOB 1'  ,  5                                                     *
/*   1 , 2 , A , 4 , B , 3 , C , 0                                       *
/*   3 , 2 , E , 4 , 0 , 0                                               *
/*   2 , 5 , D , 0                                                       *
/*   4 , 5 , G , 0                                                       *
/*   THE FIRST LINE STATES THAT  JOB 1 HAS 5 EVENTS.THE SECOND LINE      *
/*   STATES THAT EVENT 1 IS SUCCEEDED BY EVENT 2,4,AND 3 AND THE         *
/*   THE CORRESPONDING ACTIVITY TIMES ARE A,B AND C RESPECTIVELY.        *
/*   LIKEWISE,FROM THE SUBSEQUENT LINES IT IS INFERRED THAT ACTIVITY     *
/*   3-2 HAS A TIME OF E,ACTIVITY 3-4 HAS A TIME OF 0 AND SO  ON.        *
/*   SINCE THERE ARE NO ACTIVITIES STARTING FROM EVENT 5,THERE WILL      *
/*   BE NO CORRESPONDING LINE OF ENTRIES.                                *
/*   DUMMY ACTIVITIES ARE ENTERED LIKE ALL OTHER ACTIVITES,WITH AN       *
/*   ACTIVITY TIME OF 0.ACTIVITY 3--4 IN THE EXAMPLE ABOVE IS A DUMMY    *
/*   ACTIVITY.                                                           *
/*   THE ORDER OF ENTRY OF EVENTS WITHIN A LINE OR AMONG LINES IS        *
/*   INCONSEQUENTIAL.IN THE EXAMPLE ABOVE,EVENT 4 APPEARS BEFORE         *
/*   EVENT 3 IN LINE 2.ALSO, ENTRIES FOR STARTING EVENT 3 HAS BEEN       *
/*   MADE BEFORE ENTRIES FOR EVENT 2.                                    *
/*   FOR INTEGRATING THE MODULES ALSO A SIMILAR PATTERN OF DATA          *
/*   ENTRY IS USED.HOWEVER,THE NAME MUST BE ENTERED AS 'PROJECT' AND     *
/*   NO OTHER NAME MAY BE USED.FURTHER,THE ACTIVITY TIMES ARE TO BE      *
/*   REPLACED BY MODULE NAMES.EXAMPLE--                                  *
/*   'PROJECT'   ,  6                                                    *
/*   1 , 2 , 'JOB 1' , 4 , 'JOB 3' , 0                                   *
/*   AND SO ON FOR THE REMAINING EVENTS.                                 *
/*   THIS MEANS THAT THERE ARE 6 EVENTS IN THE SUPERNETWORK.EVENTS       *
/*   1 AND 2 ARE CONNECTED BY JOB 1,EVENTS 1 AND 4 BY JOB 3 ETC.         *
/*   AS IN THE CASE OF MODULES THERE WILL BE ENTRIES CORRESPONDING       *
/*   TO 5 EVENTS,ONE LESS THAN THE TOTAL NUMBER OF EVENTS.               *
/*   THE NAME OF THE MODULES HERE AND THE INDIVIDUAL MODULE DATA         *
```

```
/*  SHOULD BE IDENTICAL WITHIN QUOTES,INCLUDING BLANK SPACES.FOR       *
/*  EXAMPLE, ' JOB 1' WILL NOT BE IDENTIFIED AS 'JOB 1'.               *
/*  ALL DATA CAN BE ENTERED IN FREE FORMAT.THERE ARE NO SPECIFIED      *
/*  COLOUMNS OR FIELDS FOR ENTRY.NUMBERS MUST BE SEPARATED BY          *
/*  ATLEAST ONE BLANK OR A COMMA.NAMES MUST APPEAR WITHIN QUOTES.      *
/*  ZEROS AT THE END OF THE LISTS OF ACTIVITY TIMES ARE NECESSARY.     *
/*                                                                     *
/*                                                                     *
/*---------------------------------------------------------------------*
CONCRIP:PROCEDURE OPTIONS(MAIN);

/*  THE MAIN PROCEDURE READS THE NAMES OF THE DIFFERNT MODULES         *
/*  THE NUMBER OF EVENTS IN EACH MODULE AND FINDS THE SOLUTION TO      *
/*  THE CRITICAL PATH PROBLEM BY INVOKING THE PROCEDURE 'EXECUTE'.     *

DCL 1 MODULE(50),
      2 NAME CHAR(50) VAR,
      2 TIME FIXED;

DCL TOTAL_EVENTS FIXED;

DCL (A,B,C,I,J,K,M,P,X,Y,Z) FIXED;


M=1;
DO WHILE('1'B);
   GET LIST(NAME(M),TOTAL_EVENTS);
   CALL EXECUTE;
   IF NAME(M)='PROJECT' THEN RETURN;
   M=M+1;
END;

RETURN;

/*---------------------------------------------------------------------*
EXECUTE:PROCEDURE;

/*  THIS PROCEDURE CARRIES OUT THE ACTUAL TWO-PASS,CONVENTIONAL        *
/*  METHOD FOR FINDING THE CRITICAL PATH.THE PROCEDURE CAN BE          *
/*  DIVIDED IN TO THREE SECTIONS.THE FIRST SECTION CONSTRUCTS THE      *
/*  THE FORWARD AND BACKWARD LINKED LISTS FOR STORING THE INFOR-       *
/*  MATION ABOUT THE NETWORK.THE SECOND SECTION CARRIES OUT THE        *
/*  TWO PASSES FOR FINDING THE EARLY EVENT TIMES AND THE LATE          *
/*  FINISH TIMES BY CALLING ITS INTERNAL PROCEDURES.THE CRITCAL        *
/*  PATH IS THEN FOUND BY IDENTIFYING EVENTS WITH ZERO TOTAL FLOAT.    *
/*  THE THIRD SECTION CALCULATES THE VARIOUS FLOAT MEASURES.           *

/*  THE DEFINITION OF MOST OF THE VARIABLES ARE QUITE SELF EVIDENT.    *
/*  THE TWO STRUCTURED VARIABLES USED ARE DESCRIBED BELOW.             *
/*  A. HEADER(X) - FLINK AND BLINK IDENTIFY THE FIRST CARDS ON THE     *
/*                 FORWARD AND BACKWARD LINKED LISTS FOR THE EVENT     *
/*                 X.                                                  *
/*  B.CARD       - USED TO STORE INFORMATION ABOUT THE INTERRELA-      *
/*                 TIONSHIPS BETWEEN THE EVENTS OF THE NETWORK.        *
```

```
/*                    .EVENT STANDS FOR A SUCCESSOR OR A PREDECESSOR
/*                           EVENT Y OF EVENT X.
/*                    .TIME STANDS FOR THE DURATION OF ACTIVITY X--Y
/*                    .LINK IS USED FOR IDENTIFYING THE NEXT CARD ON
/*                          THE LINKED LIST.

DCL 1 HEADER(TOTAL_EVENTS),
      ( 2 FLINK,
        2 BLINK ) FIXED;

DCL 1 CARD(4*TOTAL_EVENTS),
      (2 EVENT,
       2 TIME,
       2 LINK ) FIXED;

DCL (LET(TOTAL_EVENTS),
     EET(TOTAL_EVENTS),
     FLAG(TOTAL_EVENTS))FIXED;

DCL SOURCE FIXED;

DCL (FREE_FLOAT,
     INDEP_FLOAT,
     SAFETY_FLOAT,
     TOTAL_FLOAT) FIXED;

DCL DIRECTION CHAR(10) VAR,
    MODULE CHAR(15) VAR;

DO I=1 TO TOTAL_EVENTS;
   LET(I),EET(I)=0;
   FLAG(I)=0;
   HEADER(I).FLINK,HEADER(I).BLINK=0;
END;

/*                                              SECTION 1 ----------------
A=0;
DO J=1 TO TOTAL_EVENTS-1;
   GET LIST(X);
   IF(NAME(M)='PROJECT' & M>1) THEN GET LIST(Y,MODULE);
   ELSE GET LIST(Y,Z);
   A=A+1;
   LOOP:;
   DIRECTION='FORWARD';
   CALL TRACE(X,DIRECTION);
   IF B=0 THEN HEADER(X).FLINK=A;
      ELSE CARD(B).LINK=A;
   CARD(A).EVENT=Y;
   CARD(A).LINK=0;
   IF(NAME(M)='PROJECT' & M>1) THEN CALL IDENTIFY(MODULE,Z);
   CARD(A).TIME=Z;
   A=A+1;
   DIRECTION='BACKWARD';
   CALL TRACE(Y,DIRECTION);
```

```
   IF(B=0) THEN HEADER(Y).BLINK=A;
     ELSE CARD(B).LINK=A;
   CARD(A).EVENT=X;
   CARD(A).LINK=0;
   CARD(A).TIME=Z;
   GET LIST(Y);
       IF (Y=0) THEN GOTO OUT;
   IF(NAME(M)='PROJECT' & M>1) THEN GET LIST(MODULE);
   ELSE GET LIST(Z);
   A=A+1;
   GOTO LOOP;
   OUT:;
END;
/*                                              --------------------------*

/*                                              SECTION 2 ---------------*
SOURCE=1;
CALL PASS(EET,FLINK);
DO I=1 TO TOTAL_EVENTS;
   FLAG(I)=0;
END;
SOURCE=TOTAL_EVENTS;
CALL PASS(LET,BLINK);
MODULE(M).TIME=EET(TOTAL_EVENTS);


PUT PAGE EDIT(NAME(M))(X(30),A);
PUT SKIP(5)EDIT('CRITICAL PATH LENGTH =',EET(TOTAL_EVENTS))
               (X(5),A,COL(40),F(4));

PUT SKIP(5) EDIT ('CRITICAL EVENTS')(X(5),A);
Y=0;
DO I=1 TO TOTAL_EVENTS;
   IF (LET(I)+EET(I)=LET(1))
      THEN DO;
              Y=Y+1;
              IF Y=8 THEN Y=1;
              PUT EDIT (I)(COL(20+Y*5),F(3));
           END;
END;
/*                                                   --------------------*

/*                                                   SECTION 3 ----------*
PUT SKIP(5)EDIT('EVENT','EVENT FLOAT')(COL(20),A,COL(35),A);
DO K=1 TO TOTAL_EVENTS;
   PUT SKIP EDIT(K,LET(1)-LET(K)-EET(K))
                (COL(21),F(3),COL(38),F(3));
END;

PUT PAGE EDIT(NAME(M))(X(30),A);
PUT SKIP(5)EDIT('ACTIVITY','TOTAL FLOAT','FREE FLOAT','INDEP. FLOAT',
               'SAFETY FLOAT')
               (COL(10),A,COL(25),A,COL(40),A,COL(55),A,COL(70),A);
```

```
DO I=1 TO TOTAL_EVENTS;
   K=HEADER(I).FLINK;
   DO WHILE(K¬=0);
      IF CARD(K).TIME=0 THEN GOTO NEXT;
      J=CARD(K).EVENT;
      TOTAL_FLOAT=LET(1)-LET(J)-EET(I)-CARD(K).TIME;
      FREE_FLOAT=EET(J)-EET(I)-CARD(K).TIME;
      INDEP_FLOAT=MAX(0,EET(J)-LET(1)+LET(I)-
                      CARD(K).TIME);
      SAFETY_FLOAT=LET(I)-LET(J)-CARD(K).TIME;
      PUT SKIP EDIT(I,'-----',J,TOTAL_FLOAT,FREE_FLOAT,INDEP_FLOAT,
                    SAFETY_FLOAT)
                   (COL(7),F(3),COL(12),A,COL(17),F(3),COL(28),F(3),
                    COL(43),F(3),COL(57),F(3),COL(72),F(3));

      NEXT:;
      K=CARD(K).LINK;
   END;
END;
JUMP:;
/*                                                   ----------------

RETURN;
/*--------------------------------------------------------------------
TRACE:PROCEDURE(EVENT,DIRECTION);

/*  THIS PROCEDURE FINDS THE LAST CARD ON TH LINKED LIST FOR THE
/*  EVENT SPECIFIED AS THE ARGUMENT.DIRECTION SPECIFIES THE FORWARD

DCL (EVENT,A)FIXED;
DCL DIRECTION CHAR(*) VAR;

IF (DIRECTION='FORWARD') THEN A=HEADER(EVENT).FLINK;
   ELSE A=HEADER(EVENT).BLINK;
IF (A¬=0) THEN DO WHILE(CARD(A).LINK¬=0);
                  A=CARD(A).LINK;
               END;
B=A;

RETURN;
END TRACE;
/*--------------------------------------------------------------------
PASS:PROCEDURE(EVENT_TIME,LINK);

/*  THIS PROCEDURE CARRIES OUT THE ACTUAL PASS.THE ARGUMENT LINK
/*  REFERS EITHER TO THE FORWARD OR TO THE BACKWARD LINKS,
/*  AND THE EVENT TIME IS EITHER THE LATEST EVENT TIME OR THE
/*  EARLIEST EVENT TIME.APPROPRIATE ARGUMENTS ARE SUPPLIED AT
/*  THE CALLING POINT DEPENDING ON THE DIRECTION OF THE PASS.

DCL LINK(*) FIXED,
    EVENT_TIME(*) FIXED,
    (A,B,DUMMY) FIXED;
```

```
A=LINK(SOURCE);
FLAG(SOURCE)=1;
DO WHILE(A¬=0);
   EVENT_TIME(EVENT(A))=EVENT_TIME(SOURCE)+CARD(A).TIME;
   A=CARD(A).LINK;
END;

J=0;
IF SOURCE=1 THEN I=2;
   ELSE I=TOTAL_EVENTS-1;
BACK:;
IF SOURCE=1 THEN DO;
                   DO WHILE(I¬>TOTAL_EVENTS-1);
                      CALL CALCULATE;
                      I=I+1;
                   END;
                   IF J=0  THEN GOTO SKIP;
                   DO J=K TO TOTAL_EVENTS-1;
                      IF FLAG(J)=0 THEN DO;
                                         I=K;
                                         J=0;
                                         GOTO BACK;
                                      END;
                   END;
                   SKIP:;
                  END;
ELSE DO;
     DO WHILE(I¬<2);
        CALL CALCULATE;
        I=I-1;
     END;
     IF J=0 THEN GOTO HOP;
     DO J=K TO 2 BY -1;
       IF FLAG(J)=0 THEN DO;
                           I=K;
                           J=0;
                           GOTO BACK;
                        END;
     END;
     HOP:;
    END;
RETURN;

/*--------------------------------------------------------------------
CALCULATE:PROCEDURE;

/*  THIS PROCEDURE CARRIES OUT THE ACTUAL CALCULATION OF THE EVENT
/*  TIMES BY ADDING THE DURATION TIME AND THE EVENT TIME OF EACH
/*  PRECEDING EVENT AND MAKING THE MAXIMAL SELECTION

   IF FLAG(I)=0 THEN CALL TEST(I);
   IF FLAG(I)=0 THEN GOTO FURTHER;
   B=LINK(I);
```

```
   DO WHILE(B¬=0);
      DUMMY=CARD(B).TIME+EVENT_TIME(I);
      IF DUMMY> EVENT_TIME(CARD(B).EVENT)
         THEN EVENT_TIME(CARD(B).EVENT)=DUMMY;
      B=CARD(B).LINK;
   END;
   GOTO CONTINUE;
   FURTHER:;
   J=J+1;
   IF J=1 THEN K=I;
   CONTINUE:;

RETURN;
END CALCULATE;
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
TEST:PROCEDURE(EVENT);

/*  THIS PROCEDURE TESTS TO SEE IF ALL THE DIRECTLY PRECEEDING (OR
    SUCCEEDING) EVENTS HAVE BEEN VISITED AND THEIR EVENT TIMES
    CALCULATED(INDICATED BY SET FLAG).IF SO THE FLAG FOR THIS EVENT
    IS ALSO SET.                                                         */

DCL (EVENT,
     A,B)  FIXED;

IF SOURCE=1 THEN A=BLINK(EVENT);
   ELSE A=FLINK(EVENT);
DO WHILE(A¬=0);
   B=CARD(A).EVENT;
   IF FLAG(B)=0 THEN GOTO AHEAD;
   A=CARD(A).LINK;
END;
FLAG(EVENT)=1;
AHEAD:;

END TEST;
/*-------------------------------------------------------------------------*/

END PASS;
/*-------------------------------------------------------------------------*/
IDENTIFY:PROCEDURE(MODULE_NAME,MODULE_TIME);

/*  THIS PROCEDURE RETRIEVES THE CRITICAL PATH TIME FOR THE MODULE       */
/*  SPECIFIED IN THE ARGUMENT.THE USER IDENTIFIES MODULES BY ALPHA-      */
/*  NUMERIC CHARACTERS BUT THE PROGRAM ASSIGNS A NUMERIC VALUE TO        */
/*  THE MODULES AND WHENEVER A MODULE IS REFERRED TO BY ITS NAME,        */
/*  A SUITABLE INTERNAL IDENTIFICATION PROCEDURE IS REQUIRED.            */

DCL MODULE_NAME CHAR(*) VAR,
    MODULE_TIME FIXED;

DO L=1 TO M-1;
```

```
   IF MODULE_NAME=NAME(L)
      THEN DO;
              MODULE_TIME=MODULE(L).TIME;
              RETURN;
           END;
END;

RETURN;
END IDENTIFY;
/*--------------------------------------------------------------------*

END EXECUTE;
/*--------------------------------------------------------------------*

END CONCRIP;
/*--------------------------------------------------------------------*

/*--------------------------------------------------------------------*
/*  THE INPUT DATA IS TO BE ENTERED NOW AFTER THE DATA CARD.          *
/*  SEE INSTRUCTIONS AT THE BEGINNING OF THE PROGRAM.                 *
*DATA
```

# VITA

Subramanian Srinivasan

Candidate for the degree of

Master of Business Administration

Report: DFSCRIP: An Implementation of a Single-pass Algorithm for the Critical Path Method

Major Field: Business Administration

Biographical:

Personal Data:

Born in Madras, India on August 23rd, 1950, son of Mr and Mrs S.Srinivasan.

Educational:

Graduated from the Birla Institute of Technology and Science, Pilani, Rajasthan, India in 1971 with a B.E(HONS) degree in Electrical Engineering. Obtained the M.E. degree, specializing in Electrical Power Systems, from the same University in 1973.

Professional Experience:

Worked in different capacities with electrical manufacturing companies in India from 1974 to 1981; as Development Engineer with Pioneer Equipment Co. Ltd., Baroda, as Assistant Development Engineer in the R&D division of Jyoti Ltd., Baroda, and as Senior Engineer with Vidyut Agni Furnaces Pvt. Ltd., Madras.