SHARE PROGRAM LIBRARY AGENCY

PROGRAM NUMBER

231003

University of Miami

1365 MEMORIAL DRIVE - CORAL GABLES, FLORIDA
(305) - 284-6257

CONTRIBUTED PROGRAM LIBRARY SUBMITTAL FORM
(for IBM S/360, 1130 and 1800)

SHARE Program Library Agency
Triangle Universities Computation Cen
P. O. Box 12076
Research Triangle Park, N. C. 27709

This form should be completed and submitted with the program package to PID at the address shown above. Standards and instructions for submitting programs are in your *User Group Reference Manual* or the *Contributed Program Submittal Standards Manual* available from PID.

(1) **Program Order Number** *(to be filled in by PID)* 360D-23.1.003

(2) **System Type** *(machine)* S/360

(3) **Search Key** . . . . . TWO-STAGE, TWO-DIMENSIONAL TRIM /II/ OR CUTTING STOCK PROGRAM

(4) **Name of Author** *(if different than submitter's)* . . . . .

(5) **Submitter's Name** *(direct technical inquiries to)* . . . . Carol E. Shanesy

(6) **Submitter's Address** . . . . . IBM NEW YORK PUBLIC SECTOR OFFICE
59 MAIDEN LANE
NEW YORK, NY 10038

(7) **Title of Program** . . . . Two-Stage, Two-Dimensional Trim Program II

(8) **Submitter's User Group Affiliation Code and Installation Code** . . . . . S PK

(9) **Submitter's Own Program Identification and Suffix** *(optional)* . . . . . TR2D R

(10) **Primary Subject Code** 23.1

(11) **Secondary Subject Codes** . . . . . 15.0 25.0 . . .

(12) **Operating or Monitor System Required** OS360

(13) **New or Revision Code** *(if revision, show prior Program Order Number in item 1)* . . . . . N

(14) **Year Completed** 69

(15) **Date of Submittal** 032569

(16) **Documentation** *(number of original pages submitted)* 0071

(17) **Abstract** *(should contain sufficient information for a reader to determine the value of the program). Listed on the reverse side of this form are subjects which may serve as a guide for a descriptive abstract.*

CONTRIBUTED PROGRAM LIBRARY SUBMITTAL FORM

Subject Guide

a. **Purpose**
b. **Programming Language used**
c. **Version and modification level or release number of IBM Programming System used, or program order number for non-IBM authored program used**
d. **Field of application**
e. **Type of routine (main program, subroutine, etc.)**
f. **Specific description of machine requirements**
g. **Engineering Changes (EC) level of equipment (if pertinent)**

ABSTRACT

Two-Stage, Two-Dimensional Trim Program II provides a linear programming solution to the two-stage two-dimensional trim or cutting stock problem. This problem can be described briefly as follows. We have a supply of material which is stocked (or produced) in one or more fixed rectangular sizes, each size having a fixed cost per unit associated with it. We also have a list of smaller rectangle sizes together with the numbers desired of each size, which are to be produced by cutting up stock-size rectangles. If any of these rectangle sizes (w x ℓ) may be cut either w or ℓ or ℓ x w, the program will take advantage of this freedom. The stock-size rectangles are cut in two stages -- first the rectangle is slit into strips with straight cuts parallel to the length edge, and then each strip is cut individually in the perpendicular direction. The cheapest way of cutting up stock to fill the orders must be determined. The program will handle up to 10 stock sizes and 50 order sizes, as presently compiled. For these dimensions, the program requires about 100,000 bytes of memory for execution. It is an independent routine, coded entirely in FORTRAN. There are no other special machine requirements beyond those for OS/360.

(Please attach additional pages if necessary) . . . . . **Total pages attached** _____

Permission to Publish

"I hereby give anyone permission to reprint, reproduce, and distribute this program to anyone else."

(18) **Signature of Submitter and Date** Carol E. Sharesy 3-25-69

(19) **Signature of Installation Addressee** Carol E. Sharesy

T4SF

TABLE OF CONTENTS

	Page No.
I. Deck Key	3
II. User Information	4
A. Detailed Program Description	4
1. Purpose	4
2. Method	5
a. Linear Programming Technique	5
b. Generation of Patterns	8
c. Starting Solution	12
3. Restrictions and Range	13
4. Precision	13
5. Program Requirements	14
6. Timing	14
B. Program Modification Aids	16
C. Input-Output Description	17
1. Input	17
2. Output	19
D. Sample Problem	23
III. Operating Instructions	28
IV. References	29
V. Systems Material	30
A. Flowcharts	30
B. List of Important Variables	42

CARD DECK KEY

Deck No. 1: FORTRAN source deck, labelled TR2DR in columns 73-77, sequenced 001 through 492 in columns 78-80, 492 cards.

Deck No. 2: Sample problem input, sequenced 001 through 022 in columns 78-80, 22 cards.

USER INFORMATION

A. Detailed Program Description

1. Purpose. TR2DR is a program for obtaining a minimum cost solution to the two-stage two-dimensional trim or "cutting stock" problem. This problem occurs in the following way: A supplier stocks (or produces) rectangles of a material in one or more fixed sizes $W_1 \times L_1$, $W_2 \times L_2$, and so on. Each stock size costs a fixed amount C_i per rectangle to produce. The supplier receives a set of orders for smaller rectangles of this material, which he fills by cutting up rectangles of the stocked sizes into rectangles of the ordered sizes by a two-stage cutting process. Each order consists of a size (width and length), the number of pieces of this size which are to be produced, and an indication as to whether the orientation of the rectangle is important. That is, if the stock has no "grain," or its direction in the ordered rectangle does not matter, so that a $w \times \ell$ rectangle may be cut either $w \times \ell$ or $\ell \times w$, the program will attempt to take advantage of this freedom. The trim problem is to fill these orders at minimum cost, or, as a special case, with a minimum area of stock wasted in trim (sizes not ordered).

Two-stage cutting means that the stock-size rectangles are cut in the following fashion: Each rectangle is first slit into strips by means of straight cuts parallel to the length edge. Each of these rectangular strips is as long as the original stock rectangle and as wide as the width* of one of the ordered rectangles. There may of course be one strip of wasted material, which could be of any width. The sum of the widths of the strips equals the width of the original stock rectangle. Next, each strip is individually cut, in the opposite direction, into rectangles. Each of these resulting rectangles is as long as one of the ordered rectangle sizes whose width is equal to or less than that of the strip. Again, there may be one rectangle wasted at the end of the strip, whose length does not equal such an ordered size. This process creates rectangles which are of the ordered sizes or slightly wider, in which case they can be trimmed to an ordered size by a single cut along the length edge.

* or the length of an ordered rectangle which may be cut in either direction. In the remainder of this discussion, "ordered rectangle width" and "ordered rectangle length" should be taken to refer not only to the sizes actually ordered but to any permissible rotations.

The solution to the problem is a series of patterns, each of which tells how to cut a stock-size rectangle into ordered sizes by the process outlined above. Associated with each pattern is a number telling how many stock rectangles should be cut in this manner, in order that the patterns collectively fill the order. The stock size to be used for each pattern is of course also indicated. In general, there are as many patterns as there are ordered rectangle sizes.

It should be stressed that the program provides a linear programming solution to this problem, and therefore the quantities to be cut of the various patterns are not necessarily whole numbers.

2. Method. The program employs a modified revised simplex method developed by P. C. Gilmore and R. E. Gomory of IBM Research. The method has been applied to one- and other two-dimensional trim problems; some of these are described in references [1] - [3].

a. Linear Programming Technique. The problem of finding a minimum cost solution to the cutting problem described in "Purpose" can be formulated as a linear programming problem, as follows:

If the stock material is available in sizes $W_1 \times L_1$, $W_2 \times L_2$, ..., and $W_k \times L_k$, and together the orders to be filled require q_1 pieces of size $w_1 \times \ell_1$, q_2 pieces of size $w_2 \times \ell_2$, and q_m of size $w_m \times \ell_m$, then the problem becomes:

(1) Minimize: $\sum_j C_{p_j} x_j$

(2) Subject to: $\sum_j a_{ij} x_j = q_i \qquad i = 1, \ldots, m$
$a_{ij} \geq 0$ and integral

where x_j is the number of units of stock to be cut up according to the j^{th} cutting pattern, p_j is the index of the stock size from which the pattern is to be cut, C_{p_j} is the cost of one unit of that stock size ($W_{p_j} \times L_{p_j}$), and a_{ij} is the number of units of the ordered size $w_i \times \ell_i$ produced each time one stock rectangle is cut according to the j^{th} pattern.

The solution to the problem is a list, each entry of which consists of a cutting pattern, the number of units of stock which should

be cut in that pattern, and the stock size to be used. In the notation above, the j^{th} such pattern is a vector $(a_{1j}, a_{2j}, \ldots, a_{mj})$ indicating that the pattern produces a_{1j} rectangles of size $w_1 \times \ell_1$, a_{2j} of size $w_2 \times \ell_2, \ldots,$ and a_{mj} of size $w_m \times \ell_m$; x_j units of stock size $W_{p_j} \times L_{p_j}$ are to be so cut. This vector, of course, does not indicate how the stock rectangle is to be cut to produce the numbers of ordered rectangles it specifies. However, that information is developed in the course of the solution and is also included in the program output.

Two problems develop in applying the techniques ordinarily applied to linear programming problems. The first is that the number of variables (that is, possible cutting patterns) is usually prohibitively great, even for "small" trim problems with relatively few order and stock sizes. This results in a matrix for the simplex method which has a very large number of columns. Apart from the difficulty of storing such a large matrix, searching through the columns at each pivot step to find the pivot column would require a great deal of time.

The second problem is that the amounts of stock $x_1, x_2, \ldots, x_j$, to be cut in the j patterns, are not generally integers, and in many applications, it does not make sense to cut a fractional number of stock rectangles into a pattern. To obtain an integral solution, these amounts may have to be rounded up or down to the next whole number. This adjustment may produce quantities of the various ordered sizes which differ slightly from the quantities ordered. Unfortunately, there is no way to guarantee a solution which, after rounding, will produce exactly the desired quantities of the ordered sizes. However, if the unrounded solution was an optimal solution to the original problem, the integer solution will be an optimal solution to the problem of obtaining the slightly different quantities produced in the rounded solution.

This program uses a form of linear programming and hence inherits the rounding problem. However, the storage and search time difficulties described earlier are overcome by using a variation of the usual simplex method. This technique is described in detail in references [1] - [3]. Briefly, it consists of treating the problem of finding an improving column (pattern) as an auxiliary problem. All possible patterns are implicitly considered, so that the solution is optimal, but only the patterns in the basis are actually stored at any given time. Consequently the storage requirement is for m basis columns, where m is the number of ordered sizes.

Since only the basis columns are stored, when the "pricing out" stage of the simplex algorithm is reached, the auxiliary problem must be solved. That is, all of the nonbasic columns must be evaluated using the current set of linear programming prices, and the best must be selected as pivot column; this is accomplished by a dynamic programming calculation. When the pivot column has been determined, it is updated to the current solution by multiplying it by the inverse of the basis matrix. The remaining steps of the simplex method -- choosing the pivot row and performing a Gaussian elimination step -- are unchanged. As in the usual simplex method, if no column can be found which will improve the current solution, then the current solution is optimal.

The "basis matrix" used in Gilmore and Gomory's method is shown below, in the notation used previously. Their method also involves the "column of constants" shown below:

Basis Matrix

$$\begin{bmatrix} 1 & -C_{p_1} & -C_{p_2} & \cdots & -C_{p_m} \\ 0 & a_{11} & a_{12} & \cdots & a_{1m} \\ 0 & a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & a_{m1} & a_{m2} & \cdots & a_{mm} \end{bmatrix}$$

Column of Constants

$$\begin{bmatrix} 0 \\ q_1 \\ q_2 \\ \vdots \\ q_m \end{bmatrix}$$

Thus, the first row of the matrix, except for the column 1 entry, represents the objective function (1). The next m rows represent the constraints (2). The constant column, excepting row 1, represents the right-hand sides of the constraint equations. And the basis columns, other than the first one, represent the patterns in the solution.

At each iteration of the solution procedure, a new variable (corresponding to a new pattern) is brought into the basis and one of the basic variables is removed. If it is the i^{th} variable which is removed, then the $(i+1)^{st}$ column is replaced by one for the new variable.

Actually, it is the inverse of the basis matrix and the product of the inverse with the constant column which are required at all times during the linear programming calculation. The entries

in the first row of the inverse (after column 1) are the linear programming prices associated with each of the ordered rectangle sizes. These prices are required for solving the auxiliary problem. The $(j+1)^{st}$ row of the product of the inverse and constant column is the number of units of stock to be cut in the j^{th} pattern, information required for determining the pivot row.

To save space, the basis matrix is not kept in storage during the computation. Instead, new patterns (columns) are saved on tape as they are generated, and a record is kept of which patterns currently correspond to which basis columns.

b. Generation of Patterns. A number of methods for generating patterns have been developed, not only for two-stage, two-dimensional cutting problems, but also other problems which lend themselves to the linear programming technique just outlined. Some of these are described in the references.

In this program, the auxiliary problem of generating a pattern is done in two stages, corresponding to the two stages of cutting: slicing the whole rectangle into strips and then cutting the strips individually. The second part is done prior to the first.

Determining the best way of cutting up a single strip of width w_j and length L is equivalent to solving the following problem:

$$\text{Maximize:} \quad \sum_i \Pi_i b_i \qquad b_i \geq 0 \text{ and integral}$$

$$\text{Subject to:} \quad \sum_i \ell_i b_i \leq L$$

where the summation is carried out over only those rectangles $w_i \times \ell_i$ for which $w_i \leq w_j$. Π_i is the linear programming price associated with the i^{th} rectangle, and b_i is the number of rectangles of size $w_i \times \ell_i$ which the strip will produce. This is a very simple type of integer programming problem called the knapsack problem, for which any number of methods of solution have been devised. Some of these are discussed in the references. This program uses the following simple technique.

As a preliminary step, the list of ordered rectangles input to the program is augmented to include rotations of those sizes which may be cut in either direction. This list is then sorted in ascending order of rectangle width, so that in determining

the best way of cutting a strip of (ordered) width w_j, only rectangles $w_i \times \ell_i$, $i \leq j$ need be considered.

A second preliminary step is to choose a grid size and express all stock and order sizes as integer multiples of this grid size. The following discussion will assume that this has been done and that all stock and order dimensions have integer values.

Now, the values for all possible strip widths ($w_1, w_2, \ldots, w_m$) and lengths ($L_1, L_2, \ldots, L_k$) are determined concurrently in the following steps:

Step 1. A "value" vector V is formed with $L_{max} + 1$ entries, where L_{max} is the length of longest stock rectangle. At any stage of the calculations, the L^{th} entry of this vector represents the value (in terms of linear programming prices) of a strip of the width under consideration and length L. L starts at 0 and runs to L_{max}. V is initialized to zeros. An index i, on the ordered widths, is initialized to 1.

Step 2. Initialize to 0 a length K which will be used to index the value vector V.

Step 3. Compute a new entry for V as follows:

$$V^*_{K+\ell_i} = \max \left[V_K + \Pi_i, \; V_{K+\ell_i} \right]$$

The new V* entry replaces the old one of the same index.

Step 4. If $K = L_j$ for any j, $j = 1, \ldots, k$, then set

$$P_{ij} = \max_{0 \leq L \leq L_j} V_L$$

This is the maximum value obtainable from a strip of width w_i and length L_j. P is called the "worth" matrix and is used in the next stage of the calculation.

Step 5. Increase the length index K by 1. If $K > L_{max}$, go to step 6. Otherwise, test

$$V_{K-1} < V_K$$

If so, return to step 3. If not, go directly to step 4. Technically, the vector V could be made monotone non-decreasing after step 5, replacing V_K by V_{K-1} if $V_K < V_{K-1}$, since clearly any way of cutting a strip L units long can be applied to one longer than L. However, no improvement is obtained from stepping off (step 3.) from a point K for which V_K is no greater than some previous point, and so these points are skipped.

Step 6. The above steps have determined the maximum values obtainable from a strip of width w_i for the various possible stock lengths. These have been saved in the i^{th} row of "worth" matrix P. The next step to increase the width index i by 1. If $i > m$, then values for all possible strips have been computed. If not, however, step 3 is repeated, and a new V vector, for widths 1 to i, is formed using the V vector for widths 1 to i - 1 as a starting point.

Having thus determined how to cut up any eligible strip into rectangles, only the other stage of the problem, that of deciding which strips to use, remains. This problem is also a knapsack problem:

(3) Maximize: $\sum_i p_{ij} d_i - C_j$ $\quad d_j \geq 0$ and integral

Subject to: $\sum_i w_i d_i \leq W_j$ for some j, $1 \leq j \leq k$

Here d_i is the number of strips of ordered width w_i into which the j^{th} stock rectangle is to be cut, C_j is the cost per unit of the j^{th} stock rectangle, W_j is the width of the stock rectangle, and p_{ij} is the value for a strip of width w_i and length L_j (from the "worth" matrix calculated in the previous stage). If no vector d and index j exist such that (3) is positive, then the current solution is optimal.

This knapsack problem is solved in somewhat the same manner as the previous one, but with some variations. Briefly, the steps used in this second stage are:

Step 1'. Again a "value" vector V is initialized to 0. In this stage V has $W_{max} + 1$ entries, where W_{max} is the width of the widest stock, and V_W represents the maximum value

obtainable from a rectangle W units wide and as long as the stock length under consideration. A second vector I of the same length as V is also initialized to zeros; it is used in step 3'.

Step 2'. A maximum current value for (3), $\bar{C}$, is initialized to $-\infty$, and a stock width index j is initialized to 1.

Step 3'. For all i, $i = 1, \ldots, m$, new entries for V are computed:

$$V^*_{K + w_i} = \max \left[V_K + p_{ij}, V_{K + w_i} \right]$$

As before, the V* entries replace the corresponding V entries and other entries are unchanged. If indeed $V_K + p_{ij}$ was greater than $V_{K + w_i}$, the index i of the length used to obtain this value is saved in the $(K + w_i)^{th}$ entry of the second vector I, which will be used in tracing which strips were used in the pattern.

Step 4'. Increase the width index K by 1. If $K > W_{max}$, go to step 5'. Otherwise again test

$$V_{K-1} < V_K$$

and return to step 3 if so. If not, repeat step 4'.

Step 5'. If

$$\bar{C} < \max_{1 \leq W \leq W_j} [V_W] - C_j$$

then replace $\bar{C}$ by $\max_{1 \leq i \leq W_j} [V_i] - C_j$ and save j at j*, the stock index on which the value $\bar{C}$ was obtained.

Step 6'. Determine which ordered widths were used in obtaining this value of $\bar{C}$ as follows:

Step 6'a. Let i* be the index such that

$$V_{i^*} = \max_{1 \leq W \leq W_j} [V_W].$$

Initialize the vector d mentioned in (3) to zeros.

Step 6'b. Let $i = I(i^*)$ and increase $d(i)$ by 1.

Step 6'c. Reduce i^* by w_i. If $i^* \leq 0$, then d has been computed. Go to step 7'. Otherwise return to step 6'b.

Step 7'. Increase stock size index j by 1. If $j > k$, this stage of the problem is complete. Otherwise, steps 3 - 7 are repeated for the new stock size.

At this point, the strips to be used in the pattern and the stock size have been determined. However, the composition of the strips is not known. This information could have been developed in the first stage of the problem by use of an index vector corresponding to the I vector of the second stage. However, the vector would have to have been saved for each ordered width. This would have necessitated the use of tape for storage, and it is quicker to essentially repeat the first stage than to save and read back this information. Consequently, the final phase of the auxiliary problem is to repeat stage 1 with the following variations:

In step 1, the value vector has only $L_j^* + 1$ entries, since the j^{*th} stock size is now the only one of interest. For the same reason, step 4 is omitted entirely. In step 3, when $V_K + \Pi_i > V_{K+\ell_j}$, the index i used to obtain the new value is saved in the $(K + \ell_j)^{th}$ position of an index vector as was done in the corresponding step of the second stage. In step 5, if $V_{K-1} \neq V_K$, the K^{th} entry of the index vector is set to zero. Finally, after step 6, if a strip of width w_i is to be used in the pattern, the composition of the strip is traced by means of the index vector. This trace is accomplished as in the other stage, except that i^* (step 6'a) is initialized to the number of the last non-zero entry of the index vector.

c. Starting Solution. Since the primal simplex method requires a feasible starting solution, or a separate phase to determine one, the following high-cost "solution" is used as a starting point.

There are m patterns in the starting solution; the j^{th} of these produces a single piece of the j^{th} ordered size. It is cut from an imaginary stock, the same size as the ordered rectangle and costing $c \cdot w_j \cdot \ell_j$, where c is a large positive number (a program constant, presently set at 100). The j^{th} pattern is cut q_j times. Hence at the start of the solution, the basis matrix is simply:

$$\begin{bmatrix} 1 & -c \cdot w_1 \cdot \ell_1 & -c \cdot w_2 \cdot \ell_2 & \cdots & -c \cdot w_m \cdot \ell_m \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & & 1 \end{bmatrix}$$

And the inverse and the product of that inverse with the constant column are:

$$\begin{bmatrix} 1 & c \cdot w_1 \cdot \ell_1 & c \cdot w_2 \cdot \ell_2 & \cdots & c \cdot w_m \cdot \ell_m \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} 0 \\ q_1 \\ q_2 \\ \vdots \\ q_m \end{bmatrix}$$

Because of the high cost of the stock used in these starting patterns, the linear programming process gradually drives them out of the solution.

3. Restrictions and Range. The program accepts up to 10 sizes of stock rectangles and up to 50 order sizes. A rectangle which may be cut in either direction counts only as a single order size. During the computation, the program considers the stock size rectangles to be divided into a grid of square units. The user specifies the size of this grid; it will usually (but not always) divide each of the widths and lengths of both stock and order sizes integrally. The number of grid units in the greatest stock dimension, either width or length, may not exceed 6399. The user may adjust some of these restrictions by recompiling (see "Program Modifications" section).

4. Precision. The program uses single precision arithmetic for the linear programming calculation. No appreciable round-off or other numerical difficulties have been experienced thus far. Both single-precision floating point and integer arithmetic are used in generating the patterns.

5. Program Requirements. This program is a conversion for System/360 of the "Two-Stage Two-Dimensional Trim Program", PK TR2D, for the 7094. The changes are very minor. It is written entirely in FORTRAN and has been compiled and tested under both the OS/360 FORTRAN G and H processors. It has been tested under Versions 14 (MFT) and 15/16 (MVT) of OS and employs no features known to the author to be peculiar to any particular version, option or modification level. The program uses one scratch data set (logical unit 2) in addition to the standard input and output (logical units 5 and 6) of FORTRAN. As presently assembled, the program requires approximately 100K bytes of storage, although this requirement can be reduced (see "Program Modifications" section). This figure does not include space for buffers for the input, output, and scratch data sets (logical units 5, 6 and 2 respectively). There are no other special machine requirements beyond those for OS/360.

6. Timing. The time required to solve a particular trim problem is very difficult to predict. It increases rapidly with the number of order sizes, and to a much lesser extent, with the number of stock sizes. The more complex the patterns (that is, the more strips per pattern and rectangles per strip, on average) the greater the run time. However, run time decreases somewhat with the number of grid units in the largest stock size, and decreases markedly as the amount of trim waste in the optimal solution increases. The following table gives some sample running times on a Model 65. The program was compiled under FORTRAN H, Option 2, for the purpose of these timing runs.

C. p. u. Time (in seconds)	No. of Order sizes plus permissible rotations	No. of Stock Sizes	No. of Iterations	Percent Waste	Complexity of Pattern: Average No. Strips per pattern	Complexity of Pattern: Average No. Rectangles per strip	Maximum Stock Dimension	Grid Size
.4	3	1	6	9.65	4.0	3.0	10 x 25	1.0
.6	6	2	9	2.67	4.0	4.5	10 x 25	1.0
.9	8	3	14	6.03	1.6	2.4	18 x 20	.5
1.0	6	1	10	28.98	1.0	1.0	115 x 230	1.0
1.2	10	1	25	26.09	2.7	2.2	40 x 25	1.0
1.7	14	2	37	13.07	2.2	1.9	20 x 40	2.0
8.0	9	1	33	9.51	10.1	7.6	48 x144	.25
11.7	22	1	101	10.65	2.1	4.1	122 x 76	2.0
14.8	32	1	50	11.12	1.1	2.4	89 x 240	1.0
40.4	32	1	78	7.34	3.2	3.2	44 x 38	.1
185.3	45	1	208	1.32	8.1	10.1	100 x 200	.5
189.9	34	6	241	2.82	4.4	4.2	50 x100	.25
191.7	52	5	62	13.90	1.2	1.9	115 x 264	.125
276.8	51	1	339	4.46	2.8	4.1	50 x 100	.25
325.0	54	6	298	2.24	3.2	3.9	50 x 100	.25
435.1	56	2	195	.77	7.7	9.8	100 x 200	.25
448.0	49	1	295	1.35	6.4	10.8	100 x 200	.25

B. Program Modification Aids

The user may wish to change the limits on the number of stock and order sizes or the maximum number of grid units permissible in the largest stock dimension. A relaxation of one restriction may require tightening of another, because of the limits of core storage.

To change the maximum number of stock sizes from 10 to N, change the dimensions of the vectors ISW, ISL, INVSL, SW, SL, and CØST from 10 to N, and change the WØRTH matrix size from 100 x 10 to 100 x N.

To change the maximum number of order sizes from 50 to N, change the dimensions of the vectors ITER, FREQ, and Y from 50 to N. Change the dimensions of RANK, ID, IW, IL, ICØN, FREQ2, FREQ1, W, L, D, and PRICE from 100 to 2 · N. Change matrix SCR1 from 100 x 3 to 2 · N x 3, matrix B from 51 x 51 to (N + 1) x (N + 1), and matrix WØRTH from 100 x 10 to 2· N x 10.

The reason for the dimensions of 2 · N in the arrays RANK, ID, IW, IL, ICØN, FREQ2, FREQ1, W, L, D, PRICE, SCR1, and WØRTH is that these arrays deal with ordered sizes and their rotations. If the user does not allow any rotations, these dimensions may be cut from 2· N to N. Or if the number of possible rotations is known to be N* < N, then they may be changed from 2· N to N* + N. Caution should be exercised however, as the input routine does not check for violation of these limits by the input data.

To change the maximum number of grid points in the greatest stock dimension from 6399 to N, change the lengths of vectors INDEX and V from 6400 to N + 1.

Finally, the constant used in the starting solution is specified at the beginning of the subroutine INPUT in the statement:

START = 100.

This constant may be increased if its present value interferes with the stock costs the user wishes to specify (see "Stock Cards" under "Input Description").

C. Input - Output Description

1. Input. The data deck for a single problem is comprised as follows:

1) Identification card
2) Grid size card
3) Stock cards (from 1 to 10)
4) Blank card
5) Order cards (from 2 to 50)
6) Blank card

Problems may be run singly or batched; if there is more than one, the data decks are simply stacked in the desired running order. The information required on each type of card is described below.

Identification card. This card simply identifies the problem on the output listing. The information is read from columns 1-30 and may consist of any alphanumeric characters anywhere in that field. The rest of the card is ignored. FORTRAN format (7A4, A2).

Grid size and iterations card. The grid size is specified in the first field, columns 1-10, of this card. It is punched with a decimal point and may appear anywhere in the field. The second field, columns 11-20, is the maximum number of iterations to be performed in attempting to find an optimal solution to the problem. When this limit is exhausted, the current solution is printed, even though it is not optimal, and the program proceeds to the next problem in the batch. If no such limit is desired, omit this field or use a value of zero. This limit is punched right adjusted in columns 11-20, with no decimal point. FORTRAN format (F10.0, I10).

The grid size is the number which is used to convert all of the stock and order dimensions to integers. It is usually chosen to be the largest quantity which will divide all dimensions evenly. However, it may be chosen somewhat larger for either of two reasons. First of all, the number of grid units in the greatest width or length in the problem may not exceed 6399. Secondly, the finer the grid, the slower the pattern generation process. The effect of choosing a grid size such that it does not divide every dimension evenly is to produce an optimum solution to a slightly different problem than the one intended. In the problem which is actually solved, every width and length not evenly divisible by the grid is increased to the next integer multiple of it.

Stock cards. Each stock rectangle size is described on a separate card. The width is punched in columns 1-10, the length in columns 11-20, and the cost per rectangle in columns 21-30. All three values should be punched with decimal points, anywhere in the respective fields. (FORTRAN format 3F10.0). Widths and lengths may be expressed in any units, but all widths, both of stock and of order

sizes, should be specified in the same units. Similarly all lengths should be measured in the same units. Widths and lengths must be expressed in the same units only if some of the ordered rectangles are allowed freedom of orientation. The costs per rectangle may be expressed in any convenient unit. However, the cost per rectangle should be considerably less than 100 times the area, because of the manner in which costs are assigned in the starting solution. If this restriction presents any difficulty, the costs can be scaled or the constant used in the starting solution may be changed (see "Program Modification Aids").

Note: To minimize the trim waste in the solution, the cost assigned to each stock size should be proportional (by the same factor for all stock sizes) to the area of the stock rectangle.

There may be from 1 to 10 stock cards. The order in which they appear in the deck is not significant.

Order cards. Each distinct rectangle size which is ordered appears on a separate card. As on the stock cards, the width appears in columns 1-10 and the length in 11-20. The number of rectangles desired of the size specified is placed in columns 21-30. These three values are punched with decimal points, anywhere in the respective fields. If a rectangle may be cut in either direction, that is, sizes $w \times \ell$ and $\ell \times w$ are equally satisfactory, then the digit 1 should be punched in column 40; if not, this field is omitted. (FORTRAN format 3F10.0, I10,) Again, the order of these cards is not significant. There must be at least two order cards and not more than 50.

Note: The input subroutine performs a few basic checks on the size of the ordered rectangles. If a rectangle clearly cannot be cut from available stock (i.e. width greater than the widest stock or length longer than the longest stock), the program will omit it in the specified form. If such an order indicates that rotation is permitted, the width and length are exchanged and the same checks are performed. If the rectangle still cannot be cut, it is omitted entirely and an error message printed. Otherwise the rotated form (only) is accepted. Likewise, an order whose rotation would not pass these checks is simply treated as an unrotatable rectangle.

Sample Problem Input. A listing of the input deck for a sample problem is shown in the next section, "Sample Problem". In this problem, there are three stock sizes from which the orders may be cut: 210 inches by 210 inches, 132 x 180, and 155 x 200. They cost

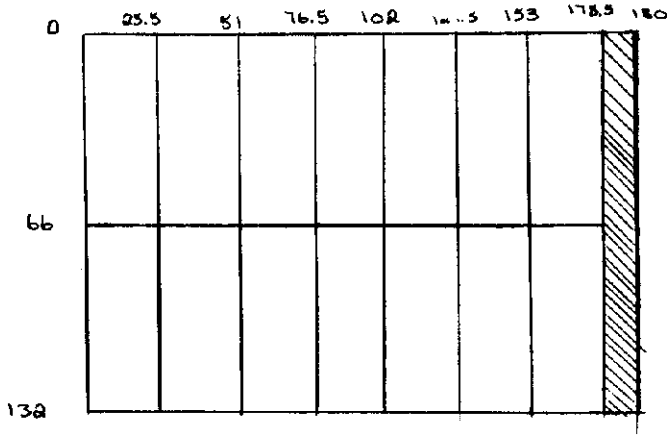
4410, 2376, and 3100 dollars per rectangle, respectively. These costs are proportional to the areas, by a factor of one-tenth, so that a minimum cost solution will also be one which minimizes the total area of stock cut (or, equivalently, the amount of trim waste in the solution). Fifteen different order sizes have been specified, of which nine can be rotated. The first size ordered is 14 inches by 36 inches; 1500 rectangles of this size are desired, and they may be cut in either direction -- 14 by 36 or 36 by 14 or some of each. The next size ordered is 15 inches by 62 inches; 1000 of this size are required, and they must all be cut 15 x 62, as no rotation is permitted. The rest of the orders are self-explanatory. The grid size has been chosen to be . 5, as that is the largest number which divides all dimensions evenly, and the quotient of the largest dimension (210) divided by the grid is only 420, well within the limit of 6399.

2. Output. The output of the program will be explained in terms of the sample problem, whose input and output are displayed in the next section: "Sample Problem."

The first section of output is simply a recapitulation of the input data. The first line of output from the program is a copy of the problem identification information provided on the identification card of the input deck. The next line states the size of the grid. Next there appears a list of the stock sizes available with appropriate headings. These are in the same form as they appeared in the input deck, except that they have been sorted in ascending order of stock width. Following this is a list of the ordered rectangle sizes, also with headings. This list includes not only the sizes specified on the input order cards, but also any permissible rotations. Rotations are indicated by the word rotation where the order quantity would ordinarily appear. The ordered sizes have also been sorted in ascending order of width.

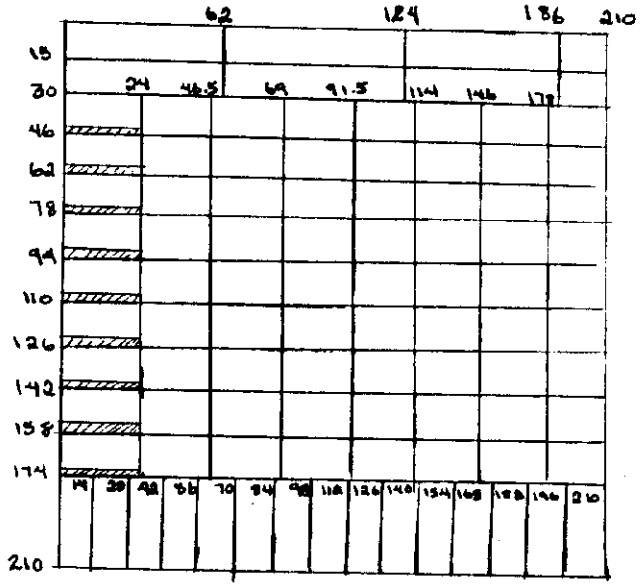
The second section of output describes the patterns in the solution, which are numbered for convenience. In the sample problem output, there are 15 patterns. The first of these is to be cut from 132. by 180. inch stock, and 142. 8571 pieces of this stock are to be cut in the pattern. For this pattern, each stock rectangle is to be cut into two strips, both 66 inches wide. Each of these strips is to be cut in to seven rectangles, all of which are 66 by 25. 5 inches. Since twice 66 inches is 132 inches, which is the width of the stock rectangle, there is no waste strip in this pattern. However, in each strip, there is a wasted rectangle, because seven times 25. 5

inches is only 178. 5 inches, 1. 5 inches less than the length of the stock. Graphically this pattern looks like this:



The shaded area represents waste.

This pattern is a very simple one; a much more interesting one is No. 6, which consists of 12 strips, of three different types. It is to be cut from 210 by 210 inch stock, and is to be repeated 28. 588 times. The first two strips are 15 inches wide, and each is to be cut into three 15 by 62 inch rectangles and one 15 by 24 piece, leaving no waste. The next nine strips, 16 inches wide, are each to be cut as follows: one rectangle 15 by 24 inches (leaving one inch waste in the width to be trimmed in that rectangle), four rectangles 16 by 22. 5 inches, and three rectangles 16 by 32 inches, again leaving no waste rectangle at the end of the strip. The remaining strip is 36 inches wide and is to be cut into 15 rectangles, all 36 by 14 inches, with no waste rectangle. This pattern appears graphically as follows:



The last portion of the output gives statistics about the solution. The first of these is the total area of stock cut by the patterns in the solution. In the notation of the "Method" section this is

$$\sum_j W_{p_j} L_{p_j} x_j$$

where j runs from 1 to the total number of patterns in the solution. The second figure given is the amount of this total area which is cut into useful sizes (i. e. ordered rectangles). In the notation used in "Method," this is

$$\sum_j x_j \left(\sum_{i=1}^{m} a_{ij} w_i l_i\right)$$

The next statistic is the number of square units of waste, which is the difference between these two figures. This figure is followed by the

percent waste, which is calculated by dividing the waste by the total area cut and expressing the result as a percentage. Finally, the number of iterations is printed; each iteration consists of generating a pattern and performing a Gaussian elimination step.

The next line of output tells whether an optimal solution was reached, as was the case in the sample problem, or whether the solution was interrupted because the maximum number of iterations, specified in the input, was executed without finding an optimal solution. If the solution was so interrupted before all the starting patterns were driven out of the solution, another section of output will appear (see "Starting Solution" under "Method"). This output begins with the message "solution incomplete (starting patterns remain)." There will follow a list of the rectangle sizes and the amounts of each size which were ordered but will not be produced by the patterns in the solution. A problem terminating in this fashion should be allowed to run for a longer period of time. This section of output will also list any rectangles which are impossible to cut from the available stock and which were not diagnosed as such in the input routine. Hence this section might even appear, under these unusual circumstances, in a run which states that the optimal solution was reached.

D. Sample Problem

Below is a listing of the input deck for the sample problem discussed earlier under "Input-Output".

```
EXAMPLE PROBLEM                                                        001
   .5                                                                  002
210.      210.      4410.                                             003
132.      180.      2376.                                             004
155.      200.      3100.                                             005
                                                                       006
 14.       36.      1500.        1                                    007
 15.       62.      1000.                                             008
 14.     23.5        700.                                             009
 22.       59.      700.                                              010
 56.       23.      650.         1                                    011
 25.       60.      175.         1                                    012
16.5       33.      750.         1                                    013
16.5       31.      125.         1                                    014
25.5       66.      2000.        1                                    015
 20.     57.5        250.                                             016
 28.       13.      300.         1                                    017
 15.       35.      1100.        1                                    018
 15.       24.      3000.                                             019
 16.     22.5       1800.        1                                    020
 16.       32.      900.                                              021
                                                                       022
```

The following is the output produced from running the sample problem. This problem required 56 seconds of c.p.u. time on a Model 65 running under OS/360, MVT, Version 15/16.

23

EXAMPLE PROBLEM

GRID= 0.5000

STOCK RECTANGLE SIZES AVAILABLE

WIDTH	LENGTH	COST PER UNIT
132.0000	180.0000	2376.0000
155.0000	200.0000	3100.0000
210.0000	210.0000	4410.0000

ORDERED RECTANGLE SIZES

WIDTH	LENGTH	QUANTITY
13.0000	28.0000	ROTATION
14.0000	36.0000	1500.
14.0000	23.5000	700.
15.0000	62.0000	1000.
15.0000	35.0000	1100.
15.0000	24.0000	3000.
16.0000	22.5000	1800.
16.0000	32.0000	900.
16.5000	33.0000	750.
16.5000	31.0000	125.
20.0000	57.5000	250.
22.0000	59.0000	700.
22.5000	16.0000	ROTATION
23.0000	56.0000	ROTATION
25.0000	60.0000	175.
25.5000	66.0000	2000.
28.0000	13.0000	300.
31.0000	16.5000	ROTATION
33.0000	16.5000	ROTATION
35.0000	15.0000	ROTATION
36.0000	14.0000	ROTATION
56.0000	23.0000	650.
60.0000	25.0000	ROTATION
66.0000	25.5000	ROTATION

EXAMPLE PROBLEM

PATTERN NUMBER	STOCK SIZE TO BE CUT WIDTH	LENGTH	NO. OF TIMES TO BE CUT	NO. OF STRIPS	WIDTH OF EACH STRIP	PIECES	COMPOSITION OF EACH STRIP WIDTH	LENGTH
1	132.0000	180.0000	142.8571	2	66.0000	7	66.0000	25.5000
2	132.0000	180.0000	33.5070	6	22.0000	3	22.0000	59.0000
3	132.0000	180.0000	38.8888	3	14.0000	1	14.0000	36.0000
						6	14.0000	23.5000
				6	15.0000	1	15.0000	35.0000
						6	15.0000	24.0000
4	132.0000	180.0000	15.2760	6	15.0000	1	15.0000	35.0000
						6	15.0000	24.0000
				1	20.0000	3	20.0000	57.5000
				1	22.0000	3	22.0000	59.0000
5	155.0000	200.0000	10.4166	2	16.0000	6	16.0000	22.5000
						2	16.0000	32.0000
				4	23.0000	2	22.5000	16.0000
						3	23.0000	56.0000
				1	31.0000	12	31.0000	16.5000
6	210.0000	210.0000	28.5879	2	15.0000	3	15.0000	62.0000
						1	15.0000	24.0000
				9	16.0000	1	15.0000	24.0000
						4	16.0000	22.5000
						3	16.0000	32.0000
				1	36.0000	15	36.0000	14.0000

7	132.0000	180.0000	17.0142	2	15.0000	1	15.0000	62.0000
						2	15.0000	35.0000
						2	15.0000	24.0000
				4	20.0000	3	20.0000	57.5000
				1	22.0000	3	22.0000	59.0000
8	155.0000	200.0000	29.6998	2	15.0000	5	15.0000	35.0000
						1	15.0000	24.0000
				2	16.5000	6	16.5000	33.0000
				4	23.0000	2	22.5000	16.0000
						3	23.0000	56.0000
9	210.0000	210.0000	5.1592	2	15.0000	3	15.0000	62.0000
						1	15.0000	24.0000
				5	36.0000	15	36.0000	14.0000
10	132.0000	180.0000	33.1502	3	14.0000	5	14.0000	36.0000
				6	15.0000	1	15.0000	62.0000
						2	15.0000	35.0000
						2	15.0000	24.0000
11	210.0000	210.0000	12.7728	14	15.0000	3	15.0000	62.0000
						1	15.0000	24.0000
12	155.0000	200.0000	21.8749	1	16.0000	6	16.0000	22.5000
						2	16.0000	32.0000
				2	16.5000	6	16.5000	33.0000
				2	23.0000	2	22.5000	16.0000
						3	23.0000	56.0000
				1	60.0000	8	60.0000	25.0000
13	155.0000	200.0000	2.7312	8	16.5000	6	16.5000	33.0000
				1	23.0000	2	22.5000	16.0000
						3	23.0000	56.0000

14	210.0000	210.0000	4.6875	2	15.0000	3	15.0000	62.0000
						1	15.0000	24.0000
				2	16.0000	1	15.0000	24.0000
						4	16.0000	22.5000
						3	16.0000	32.0000
				4	28.0000	16	28.0000	13.0000
				1	36.0000	15	36.0000	14.0000
15	155.0000	200.0000	2.4300	1	15.0000	5	15.0000	35.0000
						1	15.0000	24.0000
				3	16.0000	6	16.0000	22.5000
						2	16.0000	32.0000
				4	23.0000	2	22.5000	16.0000
						3	23.0000	56.0000

SQUARE UNITS CUT IN ABOVE PATTERNS 11009234.000

SQUARE UNITS CUT INTO ORDERED SIZES 10925861.000

SQUARE UNITS WASTED 83373.000

PERCENT WASTE IN ABOVE PATTERNS 0.757

ITERATIONS 53

SOLUTION OPTIMAL

OPERATING INSTRUCTIONS

There are no special operating instructions; all normal OS procedures are followed. The program uses the usual input and output data sets, logical units 5 and 6 respectively, and one scratch file (logical unit 2).

There is one error exit in the program, which occurs in the Gaussian elimination subroutine PIVØT. If round-off error has reached the point where no pivot row can be found, the program will exit with an appropriate message. This exit has not occurred in any test runs but is provided as a safety measure. The scratch data set used by the program is a standard FORTRAN unformatted file (logical unit 2). At each iteration of the solution algorithm, the new pattern which is to be entered into the basis in the next linear programming step is saved. Each pattern produces r+1 logical records, where r is the number of distinct (different size) "strips" in the pattern. (Strips result from the first cut and are as wide as some ordered width and as long as the stock from which they are cut.) The first of these logical records is 4 · NREC + 4 bytes long and the rest are 4 · NREC bytes, where NREC is the number of ordered rectangle sizes plus permissible rotations. The file is "rewound" at the beginning of each new problem in the batch. In testing the program, the following Job Control Language Data Definition statement was used:

```
//FT02F001 DD DCB=(RECFM=VB, LRECL=404, BLKSIZE=7294),
            UNIT=2314, SPACE=(TRK,(20, 20))
```

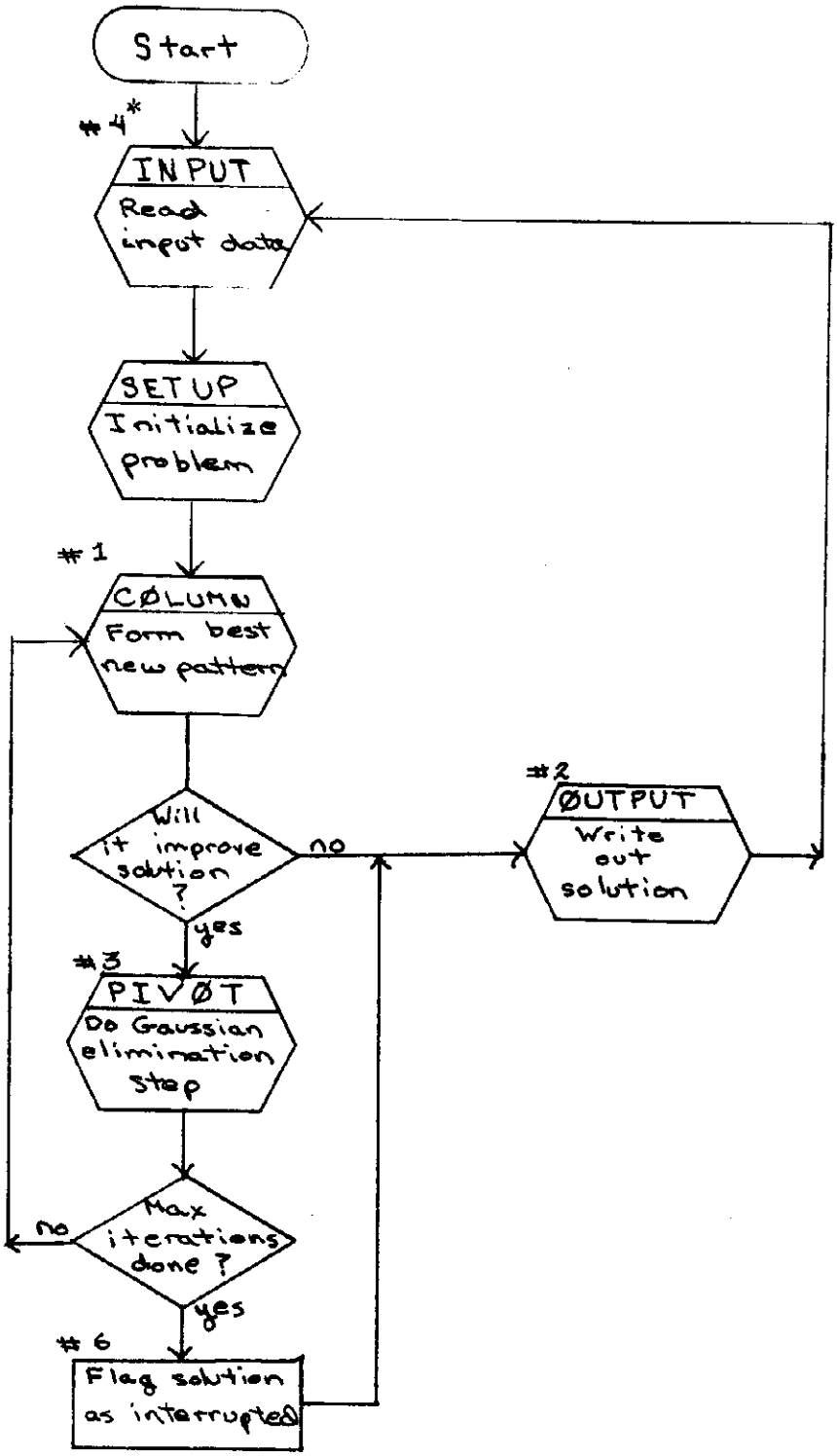
REFERENCES

1. Gilmore, P. C., and R. E. Gomory, "A Linear Programming Approach to the Cutting Stock Problem," Operations Research, Vol. 9, No. 6, November - December, 1961, 849-859.

2. Gilmore, P. C. and R. E. Gomory, "A Linear Programming Approach to the Cutting Stock Problem -- Part II," Operations Research, Vol. 11, No. 6, November - December, 1968, 863-888.

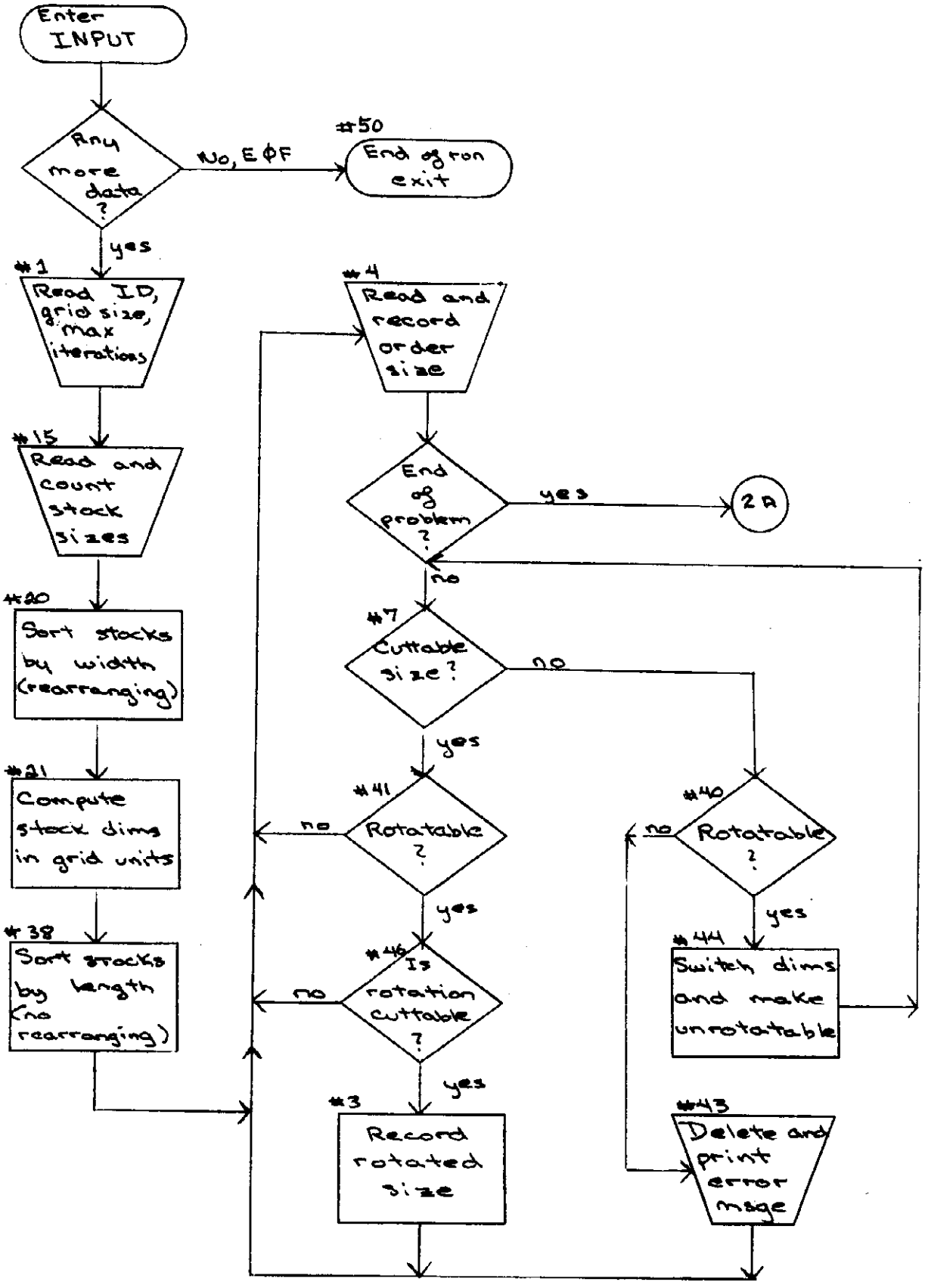
3. Gilmore, P. C., and R. E. Gomory, "Multistage Cutting Stock Problems of Two or More Dimensions," Operations Research, Vol. 13, No. 1, January - February, 1965, 94 - 119.

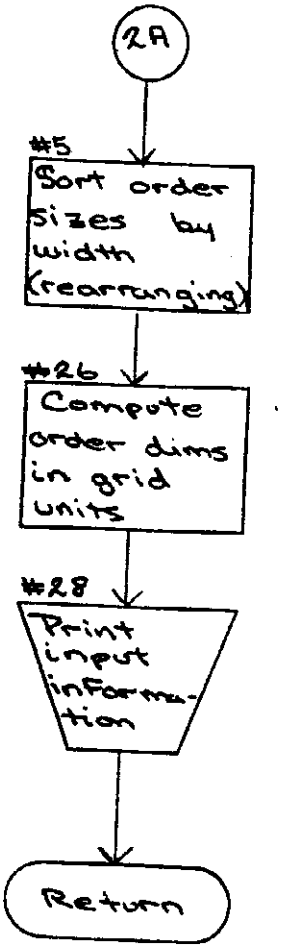
FLOW SUBROUTINE (Main Program)



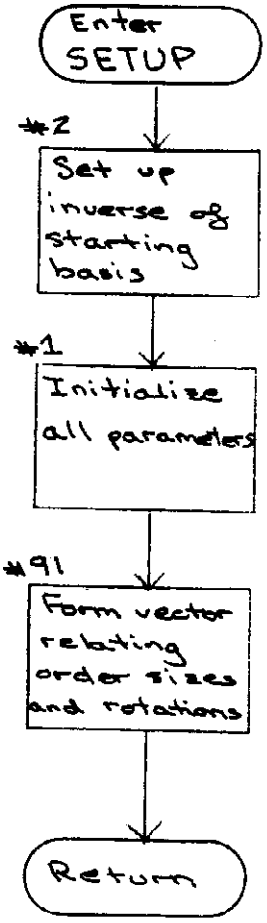
* The numbers above the boxes on the flowcharts refer to the external statements in the corresponding FORTRAN subroutines.

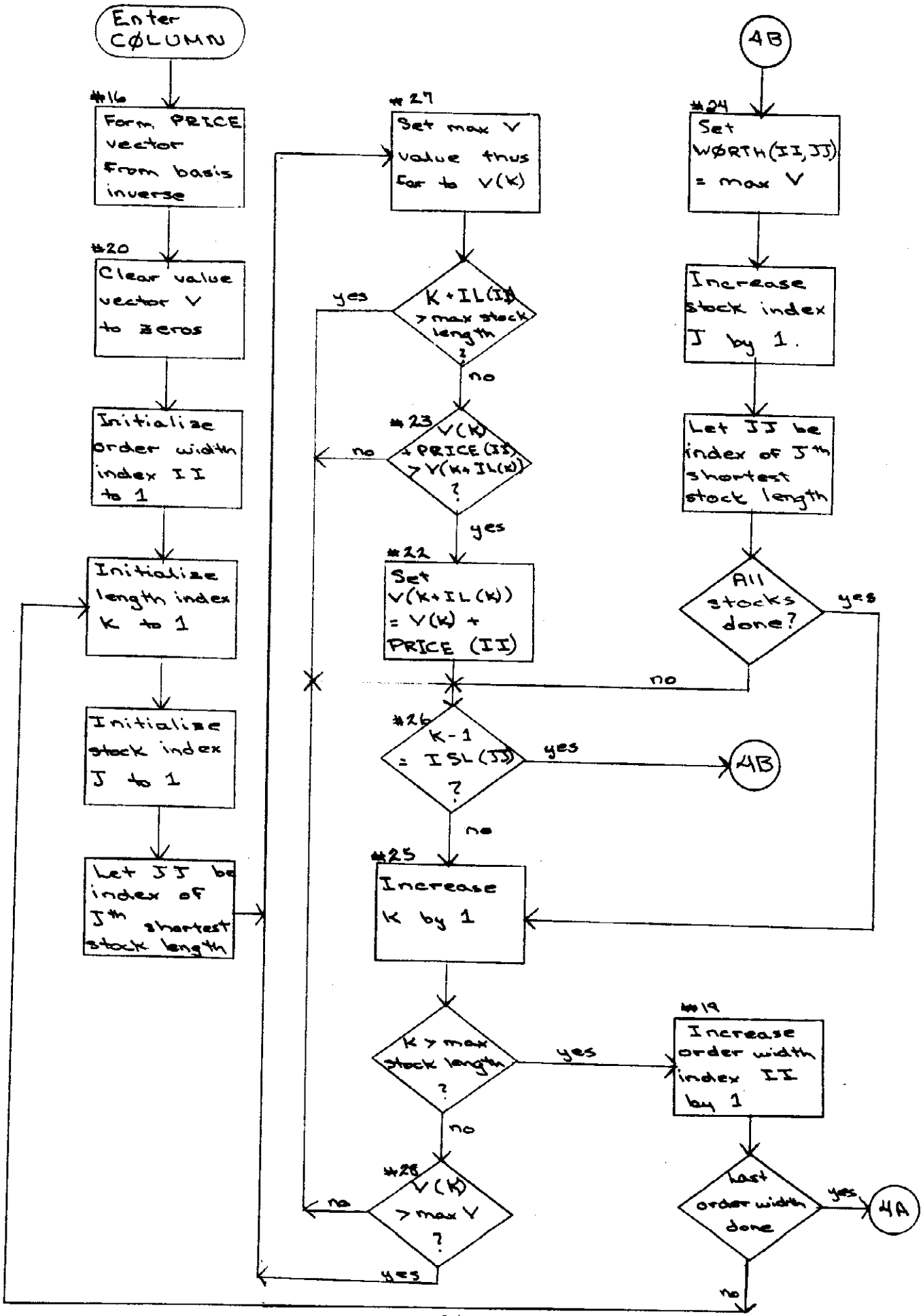
SUBROUTINE INPUT - page 1 of 2

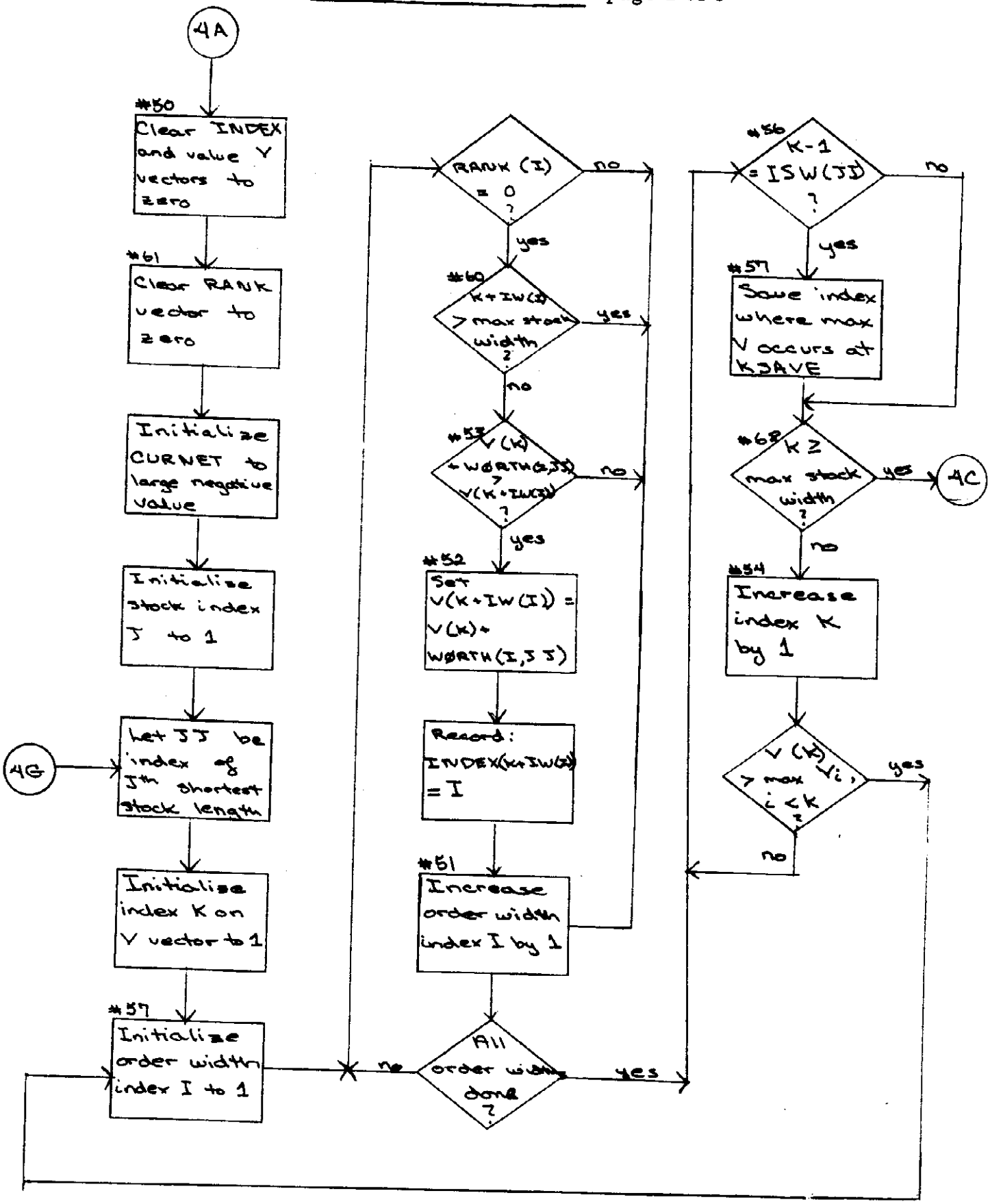


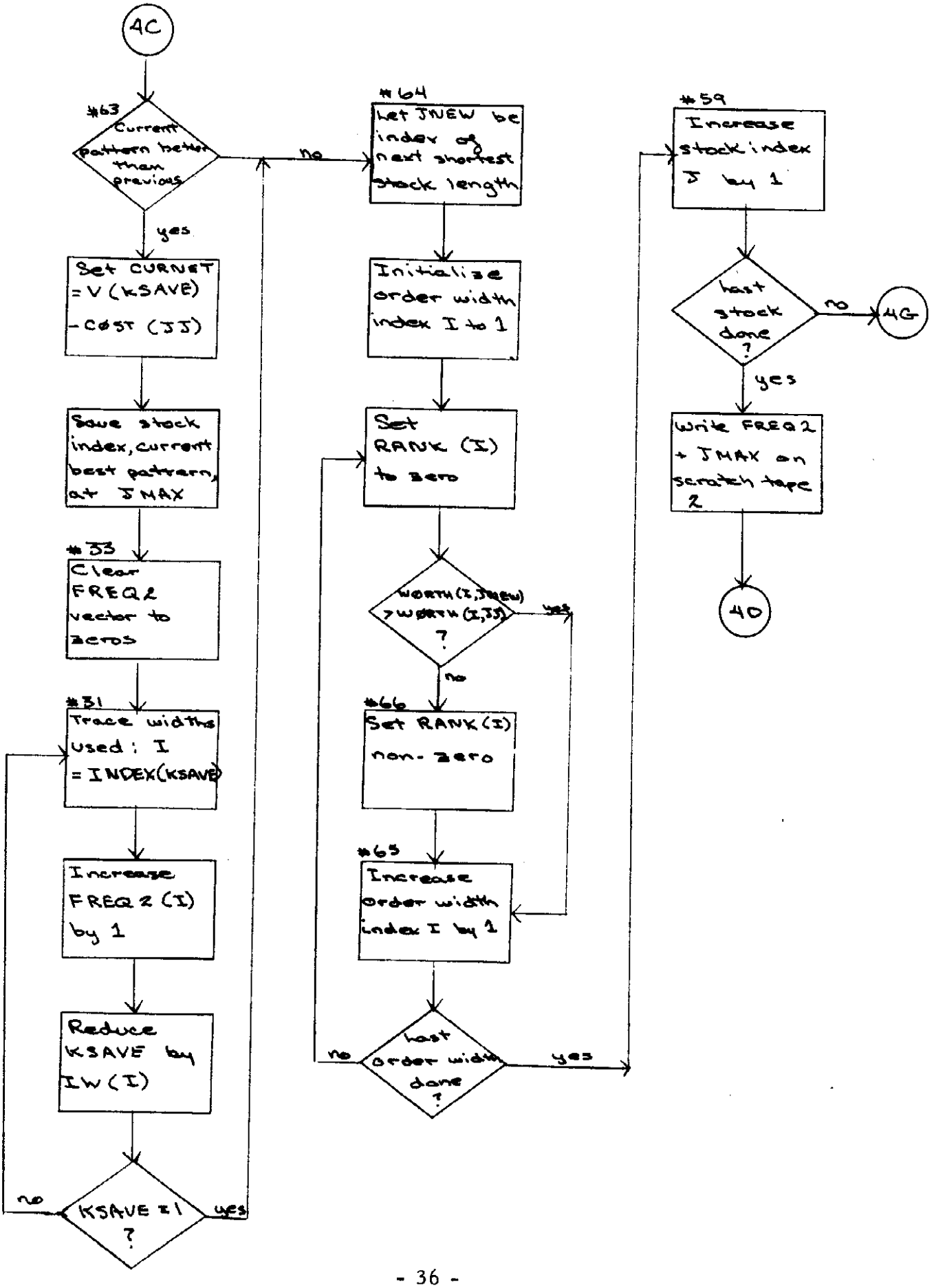

2A
#5
Sort order sizes by width (rearranging)
#26
Compute order dims in grid units
#28
Print input information
Return

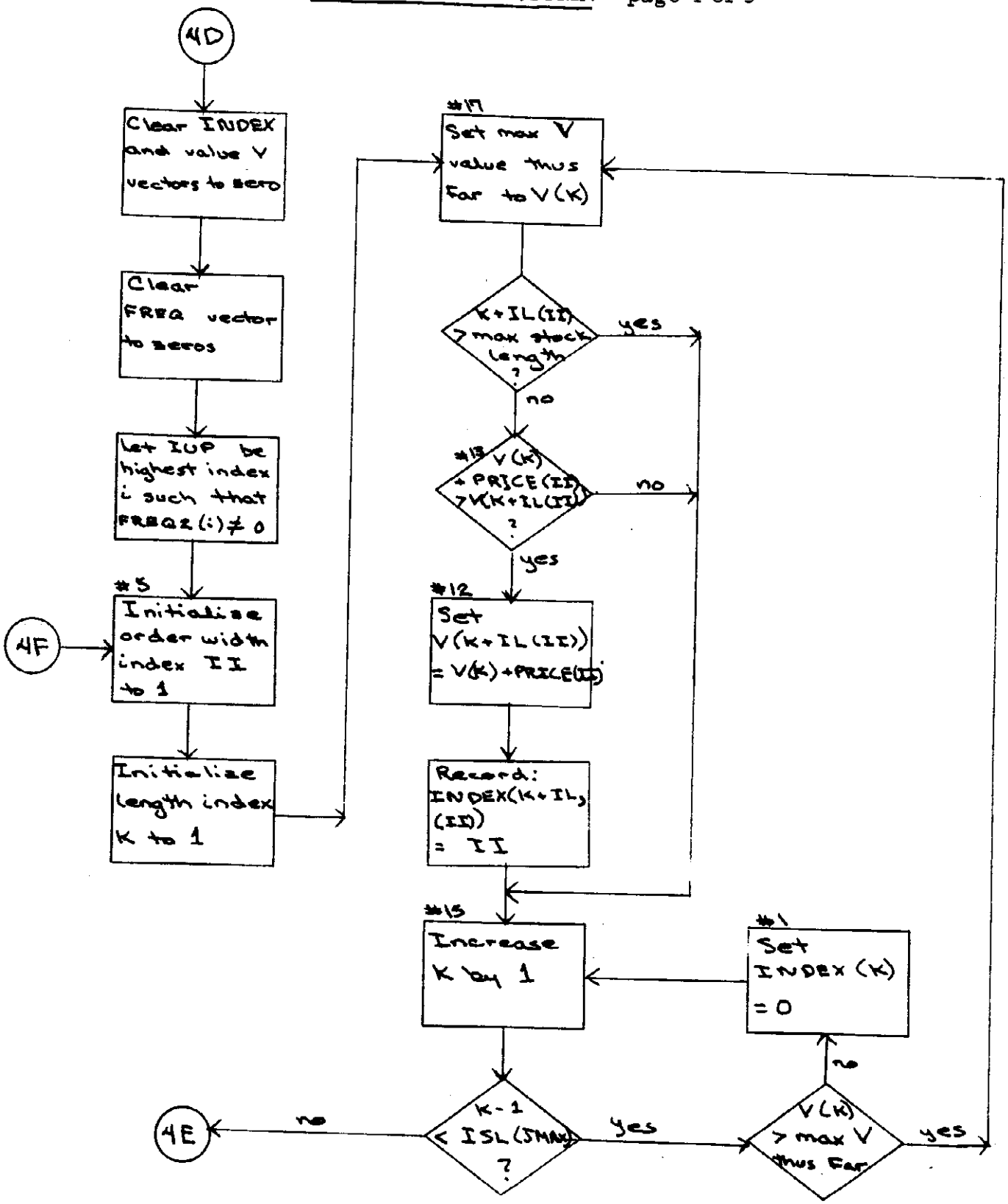
SUBROUTINE SETUP

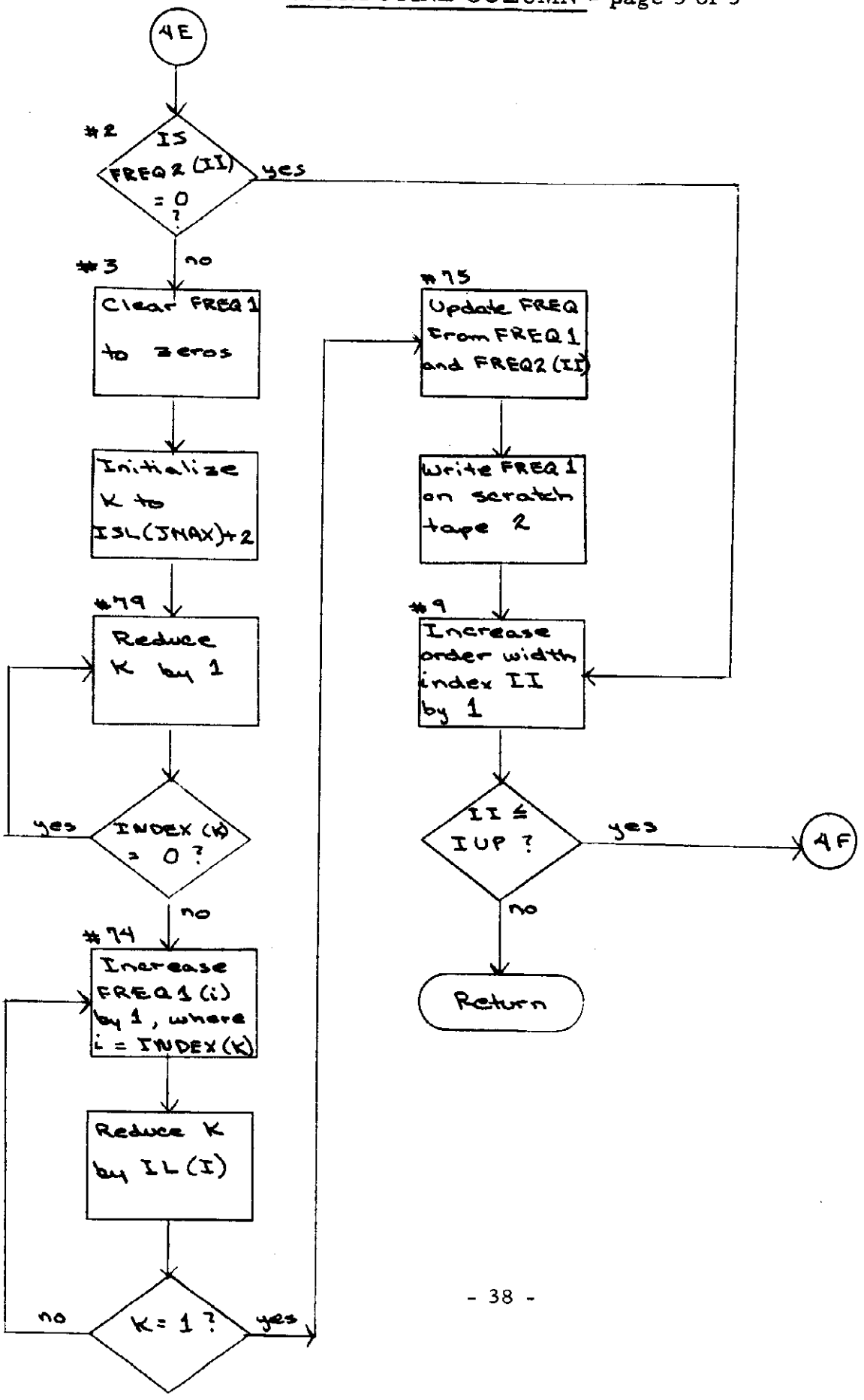



Enter COLUMN
#16
Form PRICE vector From basis inverse
#20
Clear value vector V to zeros
Initialize order width index II to 1
Initialize length index K to 1
Initialize stock index J to 1
Let JJ be index of Jth shortest stock length
#27
Set max V value thus far to V(K)
K + IL(II) > max stock length ?
yes
no
#23
V(K) + PRICE(II) > V(K+IL(K)) ?
no
yes
#22
Set V(K+IL(K)) = V(K) + PRICE(II)
#26
K-1 = ISL(JJ) ?
yes
4B
no
#25
Increase K by 1
K > max stock length ?
yes
no
#28
V(K) > max V ?
no
yes
#24
Set WORTH(II,JJ) = max V
Increase stock index J by 1.
Let JJ be index of Jth shortest stock length
All stocks done?
yes
no
#19
Increase order width index II by 1
Last order width done
yes
4A
no

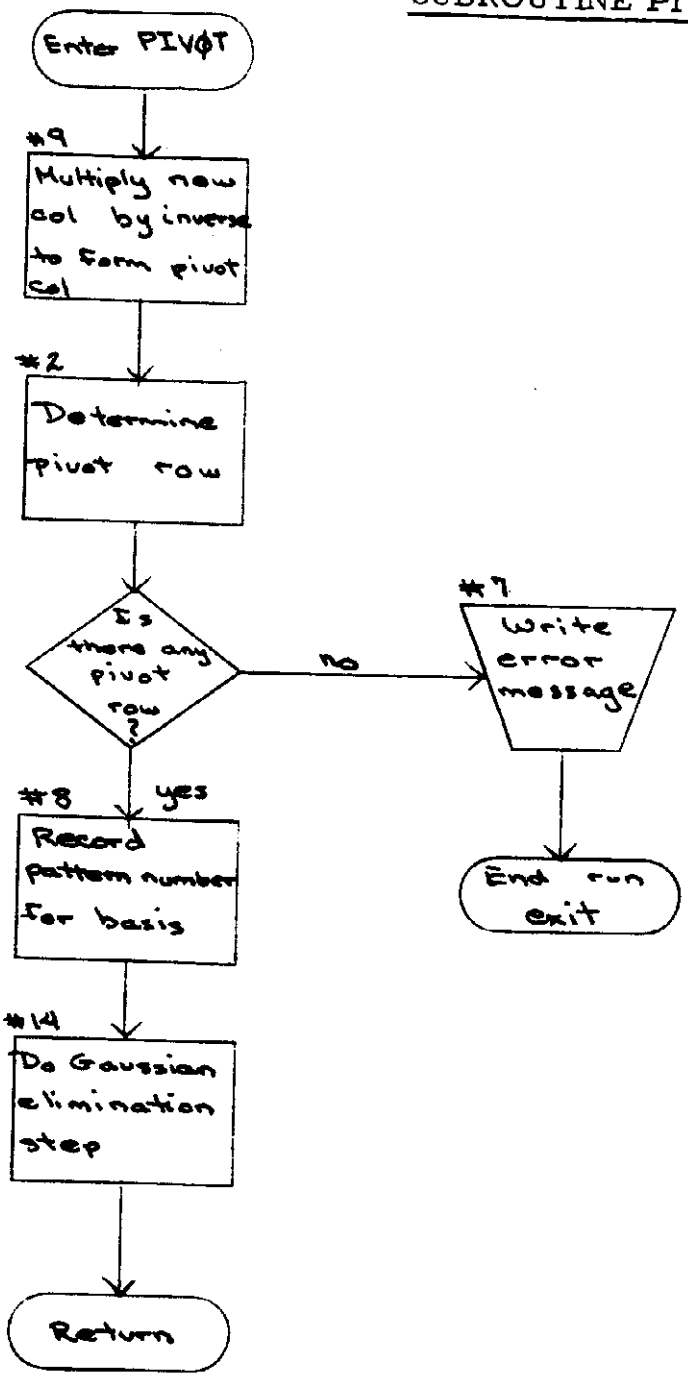

4A
#50
Clear INDEX and value V vectors to zero
#61
Clear RANK vector to zero
Initialize CURNET to large negative value
Initialize stock index J to 1
4G
Let JJ be index of Jth shortest stock length
Initialize index K on V vector to 1
#57
Initialize order width index I to 1
RANK (I) = 0 ?
no
yes
#60
K+IW(I) > max stock width ?
yes
no
#53
V(K) + WORTH(J,JJ) > V(K+IW(I)) ?
no
yes
#52
Set V(K+IW(I)) = V(K) + WORTH(I,JJ)
Record: INDEX(K+IW(I)) = I
#51
Increase order width index I by 1
All order widths done ?
no
yes
#56
K-1 = ISW(JJ) ?
no
yes
#57
Save index where max V occurs at KSAVE
#68
K ≥ max stock width ?
yes
4C
no
#54
Increase index K by 1
V(K) ≥ V(i), i < K ?
yes
no

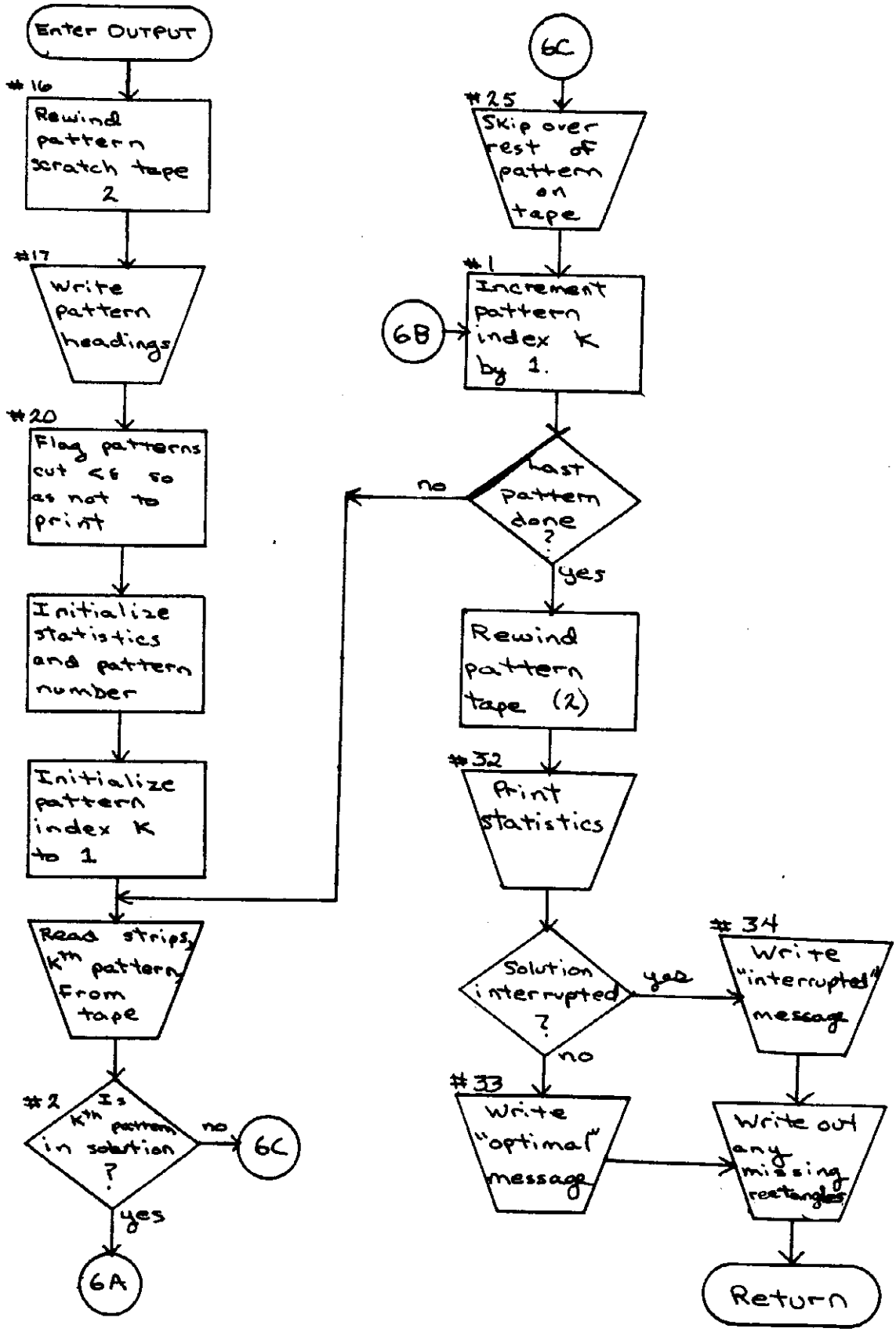

4C
#63
Current pattern better than previous
no
yes
Set CURNET = V(KSAVE) - COST(JJ)
Save stock index, current best pattern, at JMAX
#33
Clear FREQ2 vector to zeros
#31
Trace widths used; I = INDEX(KSAVE)
Increase FREQ2(I) by 1
Reduce KSAVE by IW(I)
KSAVE = 1 ?
no
yes
#64
Let JNEW be index of next shortest stock length
Initialize order width index I to 1
Set RANK(I) to zero
WORTH(I,JNEW) > WORTH(I,JJ) ?
yes
no
#66
Set RANK(I) non-zero
#65
Increase order width index I by 1
last order width done ?
no
yes
#59
Increase stock index J by 1
last stock done ?
no
4G
yes
Write FREQ2 + JMAX on scratch tape 2
40

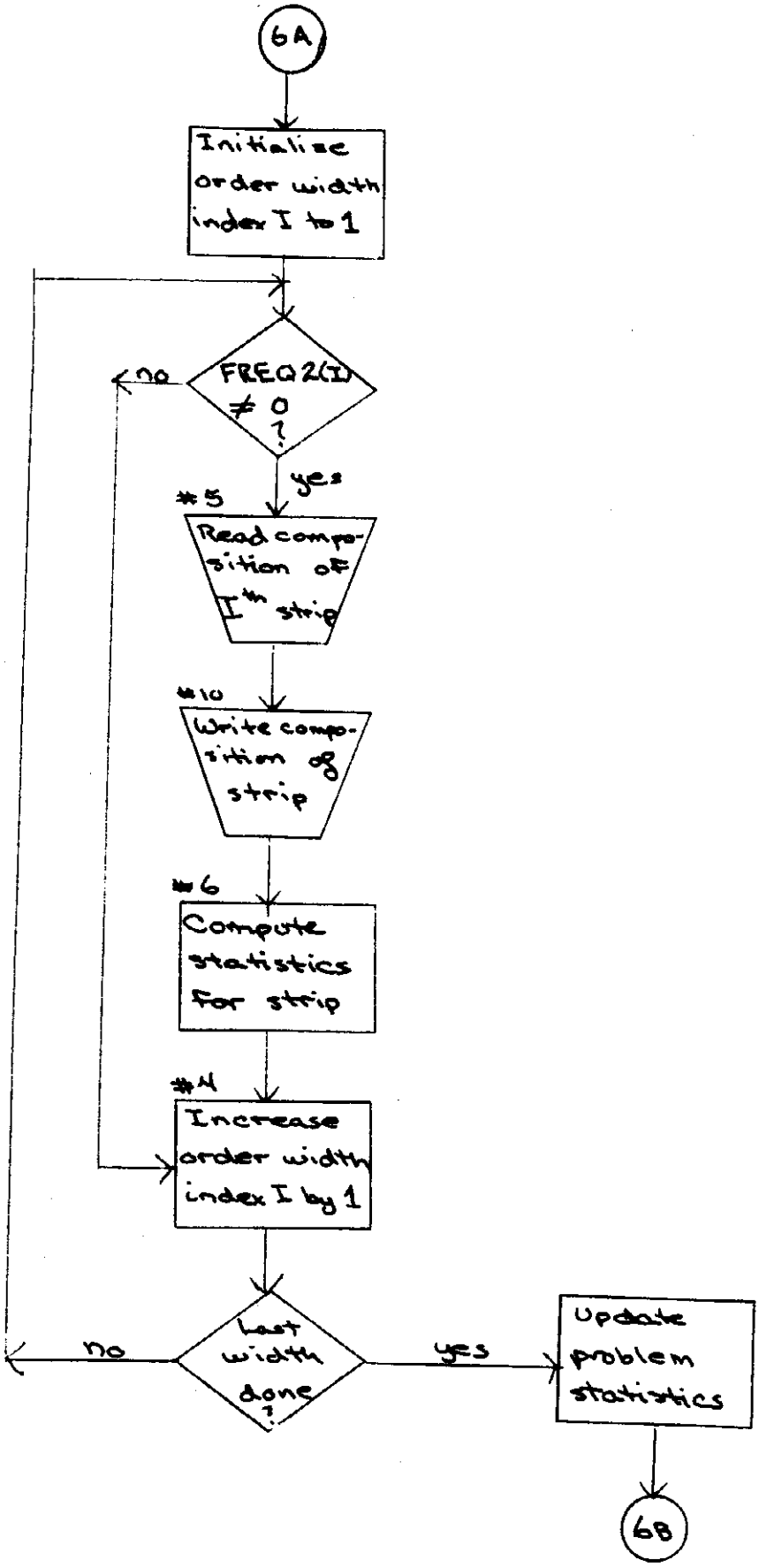

4D
Clear INDEX and value V vectors to zero
Clear FREQ vector to zeros
Let IUP be highest index i such that FREQ2(i) ≠ 0
#5
Initialize order width index II to 1
4F
Initialize length index K to 1
#17
Set max V value thus far to V(K)
K+IL(II) > max stock length ?
yes
no
#13
V(K) + PRICE(II) > V(K+IL(II)) ?
no
yes
#12
Set V(K+IL(II)) = V(K)+PRICE(II)
Record: INDEX(K+IL, (II)) = II
#15
Increase K by 1
#1
Set INDEX(K) = 0
K-1 < ISL(JMN) ?
no
4E
yes
V(K) > max V thus far
no
yes


4E
#2
IS FREQ2 (II) = 0 ?
yes
no
#3
Clear FREQ1 to zeros
Initialize K to ISL(JMAX)+2
#79
Reduce K by 1
yes
INDEX (K) = 0 ?
no
#74
Increase FREQ1 (i) by 1, where i = INDEX (K)
Reduce K by IL (I)
no
K = 1 ?
yes
#75
Update FREQ from FREQ1 and FREQ2 (II)
Write FREQ1 on scratch tape 2
#9
Increase order width index II by 1
II ≤ IUP ?
yes
4F
no
Return

SUBROUTINE PIVOT




Enter OUTPUT
#16
Rewind pattern scratch tape 2
#17
Write pattern headings
#20
Flag patterns cut <6 so as not to print
Initialize statistics and pattern number
Initialize pattern index K to 1
Read strips, Kth pattern, From tape
#2
Is Kth pattern in solution ?
no
6C
yes
6A
6C
#25
Skip over rest of pattern on tape
#1
Increment pattern index K by 1.
6B
Last pattern done ?
no
yes
Rewind pattern tape (2)
#32
Print statistics
Solution interrupted ?
yes
#34
Write "interrupted" message
no
#33
Write "optimal" message
Write out any missing rectangles
Return


6A
Initialize order width index I to 1
FREQ2(I) ≠ 0 ?
no
yes
#5
Read composition of I^th strip
#10
Write composition of strip
#6
Compute statistics for strip
#11
Increase order width index I by 1
Last width done ?
no
yes
Update problem statistics
6B

B. List of Important Variables

The following are the CØMMØN variables used in the program, which include all of the important variables.

Variable	Explanation
ALPHA	EBCDIC identification for current problem.
SW	Vector. SW(I) is the width of the I^{th} stock rectangle after the rectangles have been sorted in ascending order of width. Corresponds to W in "Method" section.
SL	Vector. SL(I) is length of the I^{th} stock rectangle after the rectangles have been sorted in ascending order of width. Corresponds to L in "Method" section.
CØST	Vector. CØST(I) is the cost per rectangle of the I^{th} stock rectangle, again after sorting by width. Corresponds to C in "Method" section.
W	Vector. W(I) is the width of the I^{th} ordered rectangle (where the list of rectangles now includes any rotations permitted), after the list has been sorted in ascending order of width. Corresponds to w in "Method" section.
L	Vector. L(I) is the length of the I^{th} ordered rectangle, again, after sorting by width. Called ℓ in the "Method" section.
D	Vector. D(I) is the number of rectangles of the size W(I) by L(I) which were ordered. This

demand was called q in the "Method" section. If the I^{th} rectangle was not an ordered size but the rotation of one, then D(I) is the negative of the index of the rectangle whose rotation it is. For this indexing purpose, used in forming ICØN, D is called ID.

ISW — Vector. ISW(I) is the number of grid units contained in SW(I). That is, SW(I)/GRID, rounded up to the next integer if not integral.

ISL — Vector. ISL(I) is the number of grid units in SL(I), i. e. SL(I)/GRID, rounded up if necessary.

INVSL — Vector. INVSL(I) is the index of the I^{th} shortest (in the length dimension) stock rectangle.

IW — Vector. IW(I) is the number of grid units in W(I), i. e. W(I)/GRID, rounded up to the next whole number if not integral.

IL — Vector. IL(I) is the number of grid units in L(I), i. e. L(I)/GRID, rounded up if necessary.

B — Matrix. Essentially the inverse of the basis matrix. The basis itself is not required at any time during the solution, until an optimum is achieved and the result is to be printed. Hence the patterns are stored on tape as they are generated and a record (ITER) is kept of those which are included in the current basis. Because the first column of the inverse (like the basis) is simply a unit vector, the product of the constant column

and the inverse is stored in the first column of B, rather than the unit vector. This column corresponds to x in the description of the "Method," and the first row of B, after column 1, corresponds to the $\underline{\Pi}$ vector of that description.

ITER — Vector. ITER(I) is the number of the iteration on which the pattern corresponding to the $(I+1)^{st}$ basis matrix column was produced.

ICØN — Vector of correspondence between input ordered rectangle sizes and the list of ordered rectangles which includes rotations. ICØN(I) is the number (less 1) of the row in the basis matrix to which the rectangle W(I) x L(I) corresponds.

INDEX — Vector. INDEX(I) is the index of the ordered rectangle last used in forming the value V(I) in pattern generation. Corresponds to the $\underline{I}$ vector in the second and third stages of that process, as described under "Method."

V — The "value" vector $\underline{V}$ formed in the pattern generating algorithm, described in "Method" section.

WØRTH — The "worth" matrix called $\underline{P}$ in the pattern generation description under "Method." WØRTH (I, J) is the greatest value obtainable from a strip of ordered width W(I) and stock length SL(J).

FREQ2	Vector. In the pattern generating subroutine CØLUMN, FREQ2(I) is the number of strips of ordered width W(I) in the current pattern. Used in the same sense in ØUTPUT subroutine. Called d in the section on "Method."
FREQ1	Vector. In the CØLUMN and ØUTPUT subroutines, FREQ1(I) is the number of rectangles W(I) by L(I) which can be cut from the strip under consideration. Corresponds to b in the pattern algorithm described under "Method."
FREQ	Vector. FREQ(I) is the number of ordered rectangles of the I^{th} smallest width (not including rotations) produced in any particular pattern. It is the vector which represents the new pattern and is entered into the basis (preceded by the appropriate row 1 entry). Corresponds to vector $(a_{1j}, a_{2j}, \ldots, a_{mj})$ described in "Method" section.
PRICE	Vector. PRICE(I) is the current linear programming price, obtained from the (ICON(I) + 1)st column of the inverse matrix, of the rectangle W(I) + L(I). Corresponds to Π in "Method" section.
NS	The number of stock rectangle sizes. Corresponds to k in "Method" section.
NREC	The number of ordered rectangle sizes plus permissible rotations of them.

NRØT	The number of ordered rectangle sizes which may be rotated.
NR	The number of ordered rectangle sizes. NR = NREC - NRØT. Corresponds to m in the "Method" section.
NR1	NR + 1. The number of rows and columns in the basis matrix.
NØ	The number of iterations the program has performed on the current problem(and the total number of patterns generated).
MAXSL	The number of grid units in the length of the longest stock rectangle. That is, the maximum of ISL(I) for I = 1, . . . , NS. Called L_{max} in "Method" section.
MAXSW	The number of grid units in the width of the widest stock rectangle. That is, the maximum of ISW(I) for I = 1, . . . , NS. Called W_{max} in "Method" section.
GRID	The grid size discussed in the "Method" and "Input-Output Description" sections.
E	A round-off control parameter.
START	The constant used in computation of costs for the starting solution patterns. Called c in the "Starting Solution" part of "Method."
CURNET	The value of expression (3) in the pattern generating procedure. Called C in the description of that procedure under "Method."

JMAX

The index of the stock rectangle size from which the current pattern is cut. Corresponds to j^* in the description in "Method".

ITMAX

The maximum number of iterations to be performed in attempting to find an optimum solution to the current problem. ITMAX=0 if the user does not wish any such cutoff. If cutoff is reached before optimum solution, ITMAX is set to -1 as signal.

CUT

The number of square units of stock material (or total area) cut by the patterns in the solution, not including starting patterns. In the notation of "Method," CUT is:

$$\sum_j W_{p_j} \cdot L_{p_j} \cdot x_j$$

j going from 1 to the total number of patterns.

USED

The number of square units of material in all the useful (ordered) rectangles produced by the patterns in the solution, again excluding starting patterns. In the notation of "Method":

$$\sum_j (x_j (\sum_{i=1}^{m} a_{ij} \cdot w_i \cdot \ell_i))$$

WASTE

The number of square units of waste material cut in the solution. WASTE = CUT - USED.

TRIM

Percentage of wasted material in patterns in the solution: TRIM = 100. *(WASTE/CUT).

Y

Vector. Y is the pivot column, updated to the current solution. It is the product of the inverse

of the basis (B, with adjustments for column l) and FREQ, the new column being brought into the solution.

RANK

Scratch vector. Used in INPUT subroutine for sorting and in CØLUMN subroutine, stage 2, for recording whether WØRTH (I, J) > WØRTH (I, J-1) for I = 1, . . . , NREC. Also used in ØUTPUT subroutine to contain the inverse of the vector ICØN.

SCR1

Scratch matrix, used in INPUT subroutine.

ID

Same as D.