and the whole unit can be worked through by the program in a few minutes.

The programmer's dream is to produce the kind of test which will cause the computer to print out "CHANGE VALVE 9 ON CHASSIS 23," but this kind of thing, while possible for a limited range of faults, can never be applied to a whole machine.

A successful set of test programs for a computer can only be built up as a result of experience of the tendencies to failure of the computer. They will contain patterns to which the computer is sensitive, and those combinations of operations and timing which it finds most difficult.

Test programs are not a certain means for diagnosing all computer ills, but they are powerful instruments which can be useful in the hands of a good engineer.

ACKNOWLEDGEMENT

The author would like to thank International Computers and Tabulators Limited for permission to publish this paper.

REFERENCES

Bird, R. (1956). "The Hec Computer," *Proc. I.E.E.*, Vol. 103B, Supplement Nos. 1–3, p. 207.
Grimsdale, R. L. (1953). "Diagnostic Programmes," N.P.L. Symposium on automatic digital computation, p. 246, *H.M.S.O.*
Wheeler, D. J., and Robertson, J. E. (1953). "Diagnostic Programs for the Illiac," *Proc. I.R.E.*, Vol. 41, No. 10, p. 1320.

Transposing Matrices in a Digital Computer

by P. F. Windley*

Summary: In November 1957 the problem of transposing a matrix in the store of a computer was given as an exercise to students taking the Cambridge University Diploma in Numerical Analysis and Automatic Computing. The best solution received was due to the author, who describes it in this paper, together with some of the other methods suggested.

Given a matrix of m (rows) by n (columns) stored by rows in a digital computer, it is required to transpose the element a_{ij} with address $ni + j$ to address $mj + i$ where $0 \leqslant i \leqslant m - 1$, $0 \leqslant j \leqslant n - 1$. If a large amount of storage space is available it is easy to place the transposed matrix elsewhere in the store and then, if necessary, transfer it back to the position it originally occupied. This is impossible with a large matrix or if storage space is limited. Methods such as that of Berman (1958) or that used in the Pegasus Matrix scheme (Ferranti, 1958) are, therefore, not considered.

One possible method to overcome this, due to M. Fieldhouse, is as follows. Suppose the element a_{00} is in location 0. Take the element a_{10} in location n, place it in location 1 and shift all the remaining elements of the first row one place further down the store. Then take element a_{20}, place it in location 2 and shift all the elements of the first two rows down one place. Repeat this process for all the elements of the first column. Consider the remaining $m(n - 1)$ elements. The first row will have been shifted $(m - 1)$ places, the second row $(m - 2)$ places, and in general the i^{th} row $(m - i - 1)$ places. Hence the element originally in $ni + j$ will now be in

$$ni + j + (m - i - 1) = i(n - 1) + (j - 1) + m, j \neq 0.$$

* Now at University of Leeds.

Consider these elements as an m by $(n - 1)$ matrix starting in location m. If this is transposed the element in $i(n - 1) + (j - 1) + m$ will be in

$$i + m(j - 1) + m = i + mj,$$

and the original matrix will have been correctly transposed. The problem has now been reduced to transposing an m by $(n - 1)$ matrix. The process can therefore be repeated $(n - 2)$ times to transpose all the elements. The whole process involves $\frac{1}{4}m(m-1)n(n-1)+(m-1)(n-1)$ reading and writing operations to and from the store. If this is being programmed for a machine with a fast working store, it may be fast for small matrices, especially if there is an instruction available which exchanges rapidly a number in the accumulator with a number in the store.

For a larger matrix it is desirable to have a method which does not involve a factor of m^2n^2 in the time required. The following method is due to J. C. Gower. Regard the transposition as a permutation of the addresses of the elements. Now any permutation may be broken down into a set of mutually exclusive cycles. Assume, at any stage, that all the cycles which contain any address between 0 and $(x - 1)$, are known to have been correctly transposed. Calculate the addresses of the cycle containing x without actually doing any transpositions. If any address occurs less than x, that cycle is known to have been transposed and a new trial can

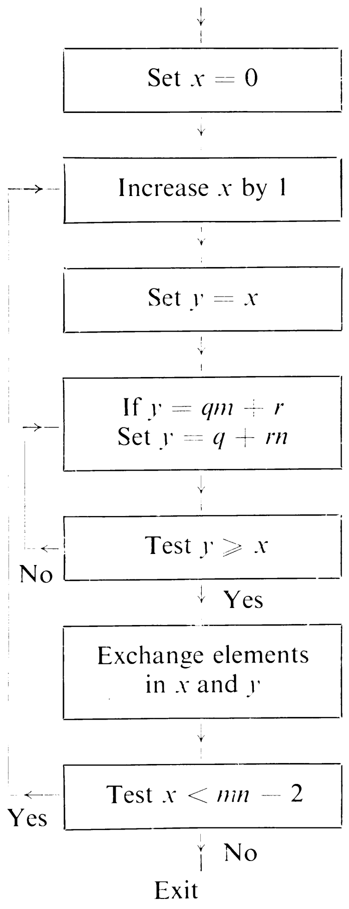

FIG. 1.—Flow diagram of the last method described.

start at $x + 1$. If the cycle returns to x without encountering any smaller address, then the cycle may be repeated and the transpositions done. To speed up the process, a count may be kept of the number of elements transposed. It is then unnecessary to test at each cycle to determine whether or not the transposition is complete. This method reduces to a minimum the number of reading and writing operations to the store.

Another method, due to the author, which saves going round each cycle once, but which involves more reading and writing operations will now be described. Consider each element in turn. Suppose that all elements to go in addresses 0 to $(x - 1)$ have been correctly positioned. At every stage place one element in its correct position, leaving those elements which are not yet correctly positioned so that they are met in the same sequence when going round a cycle. To do this start from x and calculate the addresses going backwards round the cycle containing x, without actually doing any transpositions, until the first address, y, is reached, such that $y \geqslant x$. If $y = x$ this cycle has been completely transposed, and a new trial can start at $x + 1$. If $y > x$ the element to go in x is in y. Interchange the elements in x and y. All the elements to go in addresses between 0 and x have now been transposed. The element now in y is still in its correct order, with respect to addresses greater than x, in its own cycle. The process is now repeated starting at $x + 1$. When it has been carried out for every element, the matrix will have been transposed. There is no need to consider the first element or the last two elements. The first and last element remain fixed, and the remaining element must be in its right place when all the other elements have been correctly transposed. When going round cycles it is only necessary to test for $y \geqslant x$. Exchanging element x with itself in the case of equality will be quicker and require fewer orders than to test for $y = x$ every time. The flow diagram of the method is shown in Fig. 1.

As an example consider a matrix where $m = 3$, $n = 5$.

Matrix

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix} \text{ becomes } \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix}$$

The permutation is from

(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15)

to

(1 6 11 2 7 12 3 8 13 4 9 14 5 10 15).

Or in cycle notation

(1) (2 4 10 14 12 6) (3 7 5 13 9 11) (8) (15).

In this case the method just described would cause the contents of the following pairs of addresses to be exchanged in turn:

2 and 6; 3 and 11; 4 and 6; 5 and 7; 6 and 12; 7 and 11; 8 and 8; 9 and 13; 10 and 12; 11 and 13; 12 and 14; 13 and 13.

Hence

$2 \to 6 \to 4$
$3 \to 11 \to 7$
$4 \to 6 \to 12 \to 10$
$5 \to 7 \to 11 \to 13 \to 13$
$6 \to 2$
$7 \to 5$
$8 \to 8$
$9 \to 13 \to 11$
$10 \to 12 \to 14$
$11 \to 3$
$12 \to 6$
$13 \to 9$
$14 \to 12$

REFERENCES

1. BERMAN, M. F. (1958). "A Method of Transposing a Matrix," *J. Assoc. Comp. Mach.*, Vol. 5, p. 383.
2. Ferranti Ltd. (1958). "Pegasus Matrix Interpretive Scheme," Library Program R 2500.