Inverted indexes and multi-list structures

J. Inglis

Department of Computer Science, Birkbeck College, Malet Street, London WC1E 7HX

A paper by Vose and Richardson which appeared recently in *The Computer Bulletin* proposed the use of a multi-list structure to aid the maintenance of inverted file indexes. The present paper considers this proposal further, particularly in relation to storage requirement and processing efficiency.

(Received August 1972)

In a recent paper, Vose and Richardson (1972) recommend the use of a multi-list structure in cases where frequent updating of inverted file indexes is necessary. The strategy which they describe has been suggested elsewhere—for example, by Spitzer (1969)—but does not appear to have received detailed treatment in relation to the problems of list maintenance.

Fig. 1(*b*) illustrates conventional inverted indexes (inverted lists) used in conjunction with a file consisting of the six records whose relevant contents are listed in Fig. 1(*a*); **Fig. 2** illustrates the multi-list method proposed by Vose and Richardson, applied to the same file. Further details of both methods are given by Vose and Richardson.

The present paper examines the multi-list method in greater detail, suggests some improvements, and draws attention to a number of practical considerations which are not mentioned by Vose and Richardson. In particular:

1. the need to have a master index is questioned and an alternative strategy proposed;
2. two observations are made regarding the field indexes, and an improvement proposed;
3. the storage requirement is further examined;
4. the efficiency of processing amendments and retrieval requests is examined;
5. two general observations are made regarding the multilist method.

Throughout this paper, the terminology and the variable symbols used in evaluation formulae are those defined by Vose and Richardson. The term 'multi-list method' is used to denote their proposed method rather than the use of multi-list structures generally. References to Appendix A are to Appendix A of their paper.

The master index

The evaluation of storage requirement given by Vose and Richardson assumes that a 'master index' always exists. Naturally, there will always be in the system some function to find the main-file record corresponding to a given unique identifying key, but this function need not be (and usually is not) realised in terms of a complete index. In practice, the main file is likely to be organised as a straight-forward sequential, index-sequential or random ('hashed') file, rather than as a directly-indexed file, and for these file organisations no master index exists. Nor is it necessary to employ one when inverted indexes are used—the inverted lists consist of either identifying keys or addresses; in the former case the main-file record can be located through the normal file access mechanism, and in the latter case no key translation is necessary.

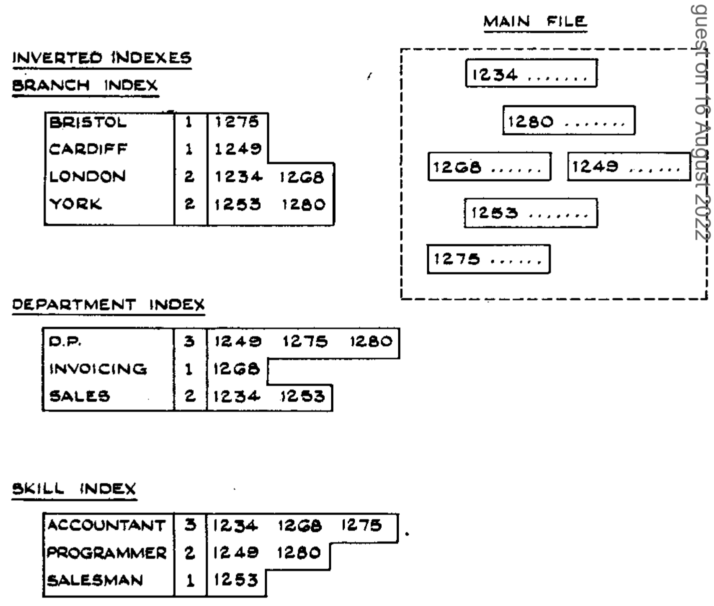
In the case of the multi-list method, though a similar argument applies, the master index has an additional function—it facilitates translation of a given identifying key to the corresponding 'sequential index number' (SIN) i.e. it fulfils the function of the 'Field-key/SIN Index' (Vose and Richardson, Fig. 6). This translation is required to locate the 'basic index' entry only when a main-file record is deleted or when the value of one of its indexed fields is amended. A new entry must be placed in the master index when a record is added to the main file.

From the values given in Appendix A(1.1), this master index is typically over 10 million bytes in length, and is held on backing store. It must be maintained in a form which makes for efficient searching for a given key, but which allows additional entries to be inserted reasonably easily. This is, of course, a familiar problem, the solution of which invariably involves a storage-space/execution-time trade off. The assumption of a tightly-packed master index in Appendix A (2 and 3) implies inefficiency in referencing the index. For an index of this size, any practical compromise will involve heavy use of storage and a number of device accesses for each reference to the index.

The storage requirement can be reduced, and the device accesses avoided altogether, by addition of a SIN field to the main-file record layout. Since every reference to the master index arises as a result of deleting a main-file record, changing

Man-no. (*key*)	*Branch*	*Department*	*Skill*
1234	LONDON	SALES	ACCOUNTANT
1249	CARDIFF	D.P.	PROGRAMMER
1253	YORK	SALES	SALESMAN
1268	LONDON	INVOICING	ACCOUNTANT
1275	BRISTOL	D.P.	ACCOUNTANT
1280	YORK	D.P.	PROGRAMMER

(*a*) **Relevant contents of six records constituting a file**



(*b*) **Inversion on Branch, Department and Skill, using inverted lists**

Fig. 1. Conventional file inversion

FIELD INDEXES

BRANCH INDEX

	L	FS	LS
BRISTOL	1	iv	iv
CARDIFF	1	v	v
LONDON	2	i	iii
YORK	2	ii	vi

DEPARTMENT INDEX

	L	FS	LS
D.P.	3	iv	vi
INVOICING	1	iii	iii
SALES	2	i	ii

SKILL INDEX

	L	FS	LS
ACCOUNTANT	3	i	iv
PROGRAMMER	2	v	vi
SALESMAN	1	ii	ii

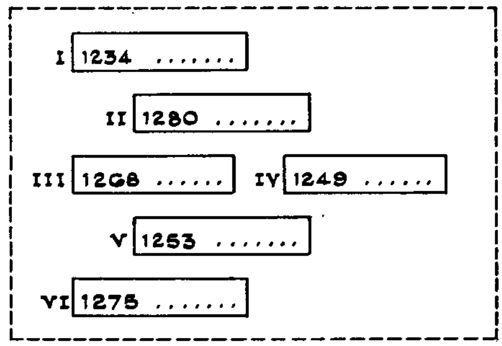
BASIC INDEX

	B	D	S	
i	iii	ii	iii	I
ii	vi	-	-	V
iii	-	-	iv	III
iv	-	v	-	VI
v	-	vi	vi	IV
vi	-	-	-	II

MASTER INDEX

1234	i	I
1249	v	IV
1253	ii	V
1268	iii	III
1275	iv	VI
1280	vi	II

MAIN FILE



Legend:

L : Length (i.e. number of entries in list)

FS : First SIN ("sequential index no.")

LS : Last SIN

B : Branch lists

D : Department lists

S : Skill lists

Fig. 2. The file of Fig. 1(*a*) organised according to Vose and Richardson's proposals (Lower-case Roman numerals denote addresses in the Basic Index (SINs), and upper-case Roman numerals addresses within the main file. Dashes (–) in the Basic Index denote null pointers.)

the value of an indexed field, or adding a new record to the main file, the main-file record is always available when the master index would be used. But it should be noted that, though discarding the master index saves storage space and makes updating more efficient, it does raise a requirement for the complete main file to be available to the basic index's housekeeping process. Housekeeping itself may be somewhat less efficient, the extent depending on the file organisation, device characteristics, and the housekeeping strategy used (for example, the changes to the main file SIN fields might be effected during a routine sequential updating run).

For the remainder of this paper, therefore, it will be assumed that (*a*) a SIN field is present in each main-file record, and (*b*) if a master index exists, it is simply part of the normal access mechanism for the main file; it is not considered in relation to operations connected with inverted indexing. **Fig. 3** illustrates a multi-list representation of the file of Fig. 1(*a*) which does not include a master index and which incorporates the treatment of field indexes described in the next paragraph.

The field indexes

1. Some reduction in the storage requirement may be achieved by removing the 'FIRST SIN' field from the field index records. This can be made possible by arranging for all linkage in the basic index to be backward, rather than forward, through the index. Such linkage has the added advantage of making the operation of adding an entry to the basic index (occasioned by creation of a new main-file record or amendment to an indexed field) considerably more efficient, since it removes the need to access previous entries in all lists to which the added record belongs. This form of linkage will be assumed throughout the remainder of this paper.

2. In theory at least, the two uses proposed by Vose and Richardson for the LENGTH field are inconsistent. If this field is used to check that all records are represented in any index, then it must reflect the number of *active* entries in the basic index list. But then it cannot be an accurate guide to the number of basic index entries to be retrieved, since *all* entries in the list (active and inactive) must be retrieved. The inaccuracy involved will be significant only when the corresponding main file field is highly volatile and/or housekeeping is infrequent.

Comparative evaluation of the methods

Assuming that it is possible to carry out unambiguously all required operations, the evaluation of a file organisation method is concerned principally with:

(*a*) the storage requirement,
(*b*) the efficiency of performing the operations of retrieval, deletion, amendment and addition of records.

In their comparison of the traditional inverted list method with the proposed multi-list method, Vose and Richardson provide an Appendix comparing storage requirements; discussion of processing efficiency is in the narrative of their paper. These two factors are now discussed further.

Storage requirement

1. The analysis for the multi-list method given in Appendix A (3) is based on the *initial* storage requirement, or the requirement immediately after housekeeping. But addition of a new record to the main file, or amendment to the value of any indexed field, causes a new entry to be generated in the basic index, while deletion or amendment of a main-file record does not remove an entry from the basic index. Thus the size of the basic index increases between housekeeping runs by (FS + A) bytes for every new record and every amendment to an indexed field.

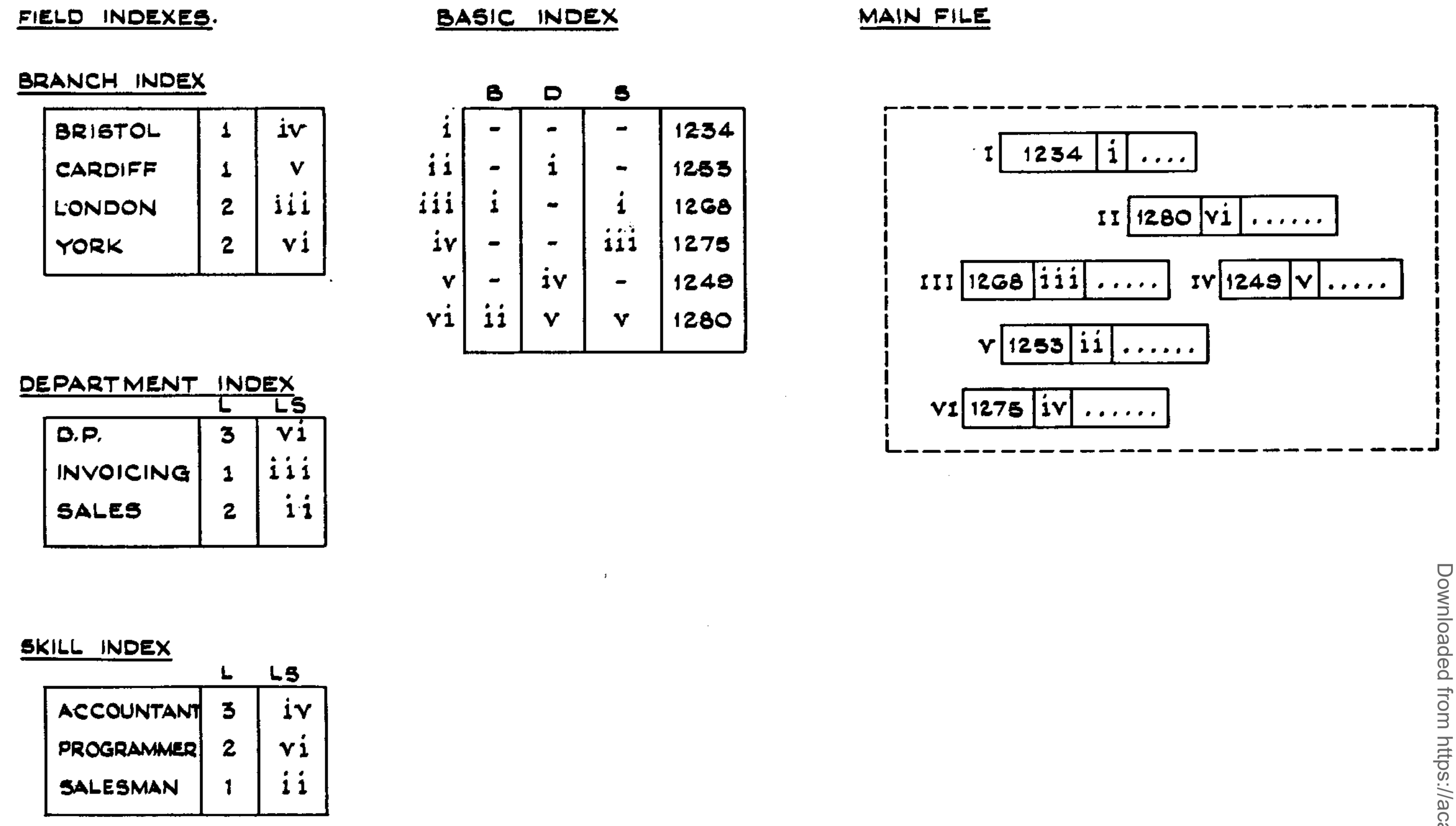


Fig. 3. An alternative organisation to that of Fig. 2.

On the assumption that inactive entries are retained also in the case of the traditional inverted list method (though removal of these is less troublesome than removal of an inactive basic index entry in the multi-list method), the corresponding increases in storage requirement between housekeeping runs under the traditional method are:

(*a*) FK bytes for every new record added to the main file, and
(*b*) K bytes for every amendment to an indexed field.

It can be verified from the typical values in Appendix A (1.1) that the increase in storage requirement on *addition* of a new record is, with minor exceptions, smaller under the multi-list method than under the inverted list method. But, under the multi-list method, the increase occasioned by each *amendment* to an indexed field can be as high as 48 bytes and always exceeds the corresponding increase under the inverted list method, whose maximum increase is eight bytes per amendment. This difference is especially significant if a file is indexed on highly volatile fields.

However, attention must be drawn to Vose and Richardson's point that addition to the basic index is a simpler operation than addition to an inverted list.

2. Vose and Richardson consciously underestimate the space required in practice for inverted lists. As they point out, if much updating of indexes is required, the variable-length lists are troublesome. In such situations, two common solutions are:

(*a*) to allocate space to each list as an integral number of fixed-length units of backing store, the store units being linked by pointers; or
(*b*) arranging storage so that several linked lists share an area of backing store.

In either of these cases, the storage requirement is greater than that taken for comparison purposes in Appendix A (2), though the initial storage requirement must be offset against the above-noted growth figures due to additions and amendments. For simplicity, the assumption of tightly-packed inverted lists will be retained in the next paragraph. An analysis of case (*a*) above, in relation to static lists, is given by Lowe (1968) and Bayes (1969).

3. The treatments of the master index and the field index already suggested involve changes to the statements of storage requirement given in Appendix A. The final expression of Appendix A (2) becomes

$$DFK + \sum_{i=1}^{F} R_i L_i$$

and the final expression of Appendix A (3) becomes

$$D[(F + 1)S + A] + \sum_{i=1}^{F} R_i L_i + (S + X) \sum_{i=1}^{F} R_i .$$

These changes do not affect the comparative evaluation of Appendix A (4). However, if the multi-list method were to include the suggested 'Field-key/SIN Index' (rather than a SIN field in the main-file records), the value of the amended expression of Appendix A (3) given above would be increased by DK, and the multi-list method would become less attractive from the point of view of storage requirement.

Processing efficiency

With the mean typical values of D, F, S and A as given by Appendix A(1.1) at 3 million, 6, $3\frac{1}{2}$ and 6 respectively, a typical storage requirement for the basic index used in the multi-list method is $D(FS + A) = 81$ million bytes. Clearly, the basic index and the field indexes are held on a backing-store device, as the inverted lists of a conventional inverted file would be. The major factor in determining the efficiency of processing operations is the number of device accesses required; this in turn depends on the block length used. Other factors determining the efficiency of processing include storage device characteristics (restrictions on block length; 'seek area' characteristics), the pattern and expected output volume of retrieval requests, and the distribution of indexed values over the basic index. It is therefore difficult to reach general conclusions regarding the efficiency of processing, but some observations may be made.

Note first that the algorithm of Vose and Richardson's Fig. 13 is intended simply to illustrate the *principle* of temporary index construction; any implementation should ensure that no block of the basic index is accessed more than once in the construction of a temporary index, i.e. that all references to a block are made before any of the lists are followed to a subsequent block. This is easily ensured by programming.

In general, for a given total buffer area in core store, the block length used for inverted lists will be smaller than that which would be used for the basic index under the multi-list method. This arises because multiple-term retrieval (e.g. 'SKILL = 2 AND DEPT = 3') requires that blocks from at least two inverted lists be in core store simultaneously, but, under the multi-list method, only one block of the basic index need be present in core store at any time. The block length used for inverted lists depends on the strategy used in multiple-term retrieval.

To obtain some measure of the situation, consider the block length for a multi-list basic index as 7000 bytes and that for an inverted list as 3500 bytes. (These appear to be realistic values.) Take the number of distinct values for an indexed field as the mean indicated by Appendix A(1.1), i.e. $R_i = 500$ for all i. Taking these in conjunction with the values of D, F, S and A (or K) used above, we derive the following typical values:

Number of entries per list (inverted or multi-list) =

$$\frac{D}{R_i} = 6{,}000 \text{ entries}.$$

Length of one entry in basic index = FS + A = 27 bytes.

Length of basic index = D(FS + A) = 81 million bytes

$$= \left\lceil \frac{81{,}000{,}000}{7{,}000} \right\rceil = 11572 \text{ blocks}.$$

Number of entries per block in basic index $= \left\lfloor \frac{7000}{27} \right\rfloor = 259.$

Main file operation	*'Read and update' operations required for indexes* — *Multi-list method*	*Inverted list method*
Add a record	(1) F Field Index entries (change LAST SIN and LENGTH)	F inverted lists, each typically several blocks long
	(2) Last block of Basic Index (insert new entry)	(add one entry to each list)
Delete a record	One Basic Index entry ('delete' the entry)	F inverted lists (remove entries or set 'deleted' values)
Amend one indexed field	(1) Two Basic Index entries, typically in different blocks ('delete' one entry; add new entry to last block)	Two inverted lists (one to 'delete' entry; one to add entry)
	(2) One Field Index entry (change LAST SIN)	

Fig. 4. Device accesses required for updating

Length of an inverted list $= \frac{DK}{R_i} = 36{,}000$ bytes

$$= \left\lceil \frac{36{,}000}{3{,}500} \right\rceil = 11 \text{ blocks}.$$

The virtue of the multi-list method is the efficiency which it brings to updating operations. **Fig. 4** of this paper compares the device accesses required for updating under the multi-list method with those required under the inverted list method; the multi-list method is clearly superior for all three operations.

For this efficiency in updating, a heavy price in retrieval performance is paid. Using the typical figures derived above, the number of device accesses (i.e. blocks read) to obtain a set of keys or addresses for a single-term retrieval criterion (e.g. 'SKILL = 2') is 11 under the inverted list method, but may be anywhere between 24 and 6000 under the multi-list method; and in the latter case low values will be obtained only when the retrieval criterion reflects 'bunching' in the basic index. On the basis of the same figures, a multiple-term retrieval criterion will involve between 24 and 11572 accesses under the multi-list method, while the maximum number of accesses required by the inverted list method rises with the number of terms, and is still measured in hundreds for a seven-term criterion.

The conclusion which must be drawn is that, where updating efficiency is more important than retrieval efficiency, the multi-list method is superior; but, if large numbers of amendments are made, the storage requirement increases rapidly. However, the multi-list method may be preferable, even from the point of view of retrieval efficiency, if the environment is such that a number of retrieval requests can be processed during a single 'pass' of the basic index. General formulation is difficult unless unwarrantable assumptions are made.

Further considerations

Finally, two important features of the multi-list method should be mentioned:

1. The multi-list method is based on the assumption, regrettably common in the literature of file-handling, that an attribute may have only a single value in a main-file record. If the straightforward OCCURS clause of COBOL is appropriate in the description of the record, then each 'occurrence' must be regarded as a distinct field, and has its own field index and its own set of linked lists in the basic index. This leads to inefficient storage and to artificially-expanded retrieval criteria, exemplified by the expansion of 'DEPT = 2' to 'DEPT(1) = 2 OR DEPT(2) = 2 OR . . . OR DEPT(N) = 2', where there are N occurrences of the field DEPT. In contrast, a traditional inverted index for an attribute may contain the same record key in the lists associated with several values and the retrieval criterion need not be expanded.

   The situation is worse in the case of a field with a variable number of occurrences, such as would be specified by the use of OCCURS DEPENDING in COBOL. The multi-list method, as described by Vose and Richardson, requires that fields exist in every basic index entry for the maximum number of occurrences. In the circumstances under which a main file record would be designed with a variable number of occurrences of a field, the wastage of space in the basic index would be considerable. This problem can, however, be overcome by a relatively minor change in the method.
2. Retrieval criteria containing 'NOT' (e.g. 'SKILL NOT = 2') are, in the absence of a master index, more easily, and in general more efficiently, provided for when the multi-list method is used.

Acknowledgement

The author is grateful to Dr. J. J. Florentin for valuable suggestions on the presentation of this paper.

References

BAYES, A. J. (1969). Retrieval times for a packed direct access inverted file. *CACM*, Vol. 12, No. 10, p. 582.
LOWE, T. C. (1968). The influence of data base characteristics and usage on direct access file organisation. *JACM*, Vol. 15, No. 4, p. 535.
SPITZER, J. H. (1969). Storing the directory for an inverted list system. *Data Base*, Vol. 1, No. 4, p. 12.
VOSE, M. R., and RICHARDSON, J. S. (1972). An approach to inverted index maintenance. *The Computer Bulletin*, Vol. 16, No. 5, p. 256.

Book reviews

The Computer in Design, by A. Hyman, 1973; 112 pages. (*Studio Vista Publishers*, £2·25)

The aim of this book is to present to art students and designers the possibilities for using computers as tools for design. A noble aim indeed, for techniques in computer-aided design (CAD) already developed have not yet been tried out in earnest outside the technological environment. Artist-designers would undoubtedly want to further develop the techniques to meet their particular needs.

No knowledge of computing is assumed by the author who starts with his first chapter entitled 'What is a computer'. He goes on to review peripherals that draw pictures, to mention problems of handling three-dimensional geometries, to discuss technical and aesthetic aspects of design, and finally to describe CAD systems, taking as an example the bottle design program developed at the CAD Centre, Cambridge. Approximately half the book is illustrations, with rather more computer generated shaded pictures than are necessary to make the point that they are possible to produce.

Success has been achieved in making the book non-technical and devoid of jargon, but, alas, it is trivial. The author has failed to distil sufficient of the basic ideas of the subject, and, perhaps more importantly, the many remaining unsolved problems, and to present them in a logical order. Nor has he been rigorous with his definitions, for example: 'A device is said to be on line if it can be used without someone physically lifting say a reel of magnetic tape from one place to another'. The best the book can hope for is to stimulate readers to seek information on the subject elsewhere. An excellent place to start would be the book of similar size by Negroponte, *The Architecture Machine* (MIT Press, 1970, £2·95) which is itself stimulating.

C. A. LANG (Cambridge)

Numerical Analysis of Symmetric Matrices, by H. R. Schwarz, H. Rutishauser and E. Stiefel, translated by P. Hertelendy, 1972; 276 pages. (*Prentice-Hall Inc., Englewood Cliffs*, £7·25)

This book, apart from minor additions, is the English translation of the 1968 German edition (reviewed in *The Computer Journal*, Vol. 14, No. 4, 1971), which developed from material prepared for a series of lectures at the Federal Technical Institute in Zurich. The book assumes familiarity with the elements of linear and matrix algebra together with some knowledge of numerical analysis.

The first chapter of the book (37 pages) deals with linear vector spaces, matrix norms and the direct numerical solution of symmetric definite systems of equations. Relaxation methods are dealt with in Chapter 2 (38 pages) which is introduced by a discusion on the connection between a symmetric definite system and a variational problem. The Gauss Seidel, S.O.R. and gradient methods are discussed in some detail. Chapter 4 (34 pages) is concerned with the least squares data fitting problem with and without constraints. The problem of poor conditioning of the normal equations is clearly discussed and illustrated with an example; methods for overcoming this difficulty are described. Chapter 4 (94 pages) is devoted to symmetric eigenvalue problems, and gives an account of the methods of Jacobi, Givens, Householder and of vector iteration methods. This is followed by a section on the LR transformation and the QD algorithm and the chapter concludes with a very welcome 'best buy' summary of the techniques described earlier. The fifth and final chapter (45 pages) is devoted to boundary value problems and the relaxation methods employed for their solution. Here the connection between self adjoint variational problems and their resulting symmetric discretisation is described. A section of this chapter considers the ADI schemes.

Examples are used throughout the book to illustrate the text, and a useful effort is made to give geometrical interpretations of some of the topics. The book contains a number of problems (which were not included in the German edition) that help the reader absorb the material and which introduce additional points.

An attractive feature of the book is the liberal inclusion of ALGOL programs of the described algorithms; FORTRAN versions of these are given in an appendix. The book as a whole is impressive; the treatment of the material is sound and the text is as readable as could be expected considering the amount of material included in a volume of this size. The book will be of direct interest to the numerical analyst and will provide engineers and scientists with a useful work of reference.

D. H. FERRISS (Teddington)

Data Processing Systems Design, by H. D. Clifton, 1971; 150 pages (*Business Books*, £4·00)

Despite its title this is not a textbook on Systems Design, but rather a set of case studies from the application of computers to business as is stated on the inside sub-heading. To obtain full benefit from it and appreciate all the references, it needs to be read in conjunction with its companion book *Systems Analysis for Business Data Processing*.

The selection of eight case studies is of interest to management of manufacturing concerns but the level of detail shown is rather inconsistent; for example there is considerable detail on input, less detail on processing and output, and practically nothing on accuracy control, error correction, and the method of determining requirements. The case studies provide little of interest to anyone in a purely commercial business (banking, insurance, building society, etc.) or in administrative organisations.

The aim in reading such a book must be either to get the general idea of systems or to pick up one or two ideas for incorporation into other systems. The bibliographical references after each case study are extensive and useful.

The standardisation of documentation between case studies and the use of an accepted standard (e.g. NCC) would have been helpful.

A book that is worth reading for the ideas it may provoke.

A. J. THOMAS (Kingston)