A Generalized Partial Pass Block Sort

A. BAYES
IBM Australia Limited, St. Kilda, Victoria, Australia

The design of a partial pass block sort with arbitrary range of key and number of work files is described. The design is a generalization of the Partial Pass Column Sort by Ashenhurst and the Amphisbaenic Sort by Nagler. The power of the sort is tabulated for various sizes of input file and number of work files. Consideration is given to the problem of combining a block sort with internal sorts, and to the best use of direct access storage devices.

KEY WORDS AND PHRASES: block sort, partial pass sort, direct access devices, column sort, chaining, reverse chaining, sort, amphisbaenic
CR CATEGORIES: 5.31

Background and Definitions

In this paper the word file is used to mean a serial file such as a magnetic tape file or a sequential file on disk.

The range r of a key is defined to be the number of distinct values that the key can take. The definition does not preclude character keys, though for ease of presentation further discussion will be restricted to contiguous numeric keys.

A block sort is a classification of records according to the value of a key associated with each record. A column sort is a particular form of block sort in which an r-way classification depends on the r possible values of a character (or column) in the key.

The amphisbaenic sort is a particular arrangement of a repeated column sort employing partial passes. Given $b + 1$ files the key is converted to a base b number. The sort proceeds as a sequence of classifications on a digit position, allocating digits to the available output files according to the rank of the file index. A simple merge of the last b subblocks occurs after a sort on the lowest column. The files are written forward and read backward without intervening rewind. The block chosen at each step is the one with the lowest numbered keys. Appendix 2 shows an amphisbaenic sort. Regarding the key as a base 4 number, there is a classification on the high order digit from file 0, followed by a classification on the low order digit and a merge to file 0 for each of the subblocks. The amphisbaenic sort is described in more detail in Nagler [3] and in Iverson [2 p. 195].

The Ashenhurst partial pass column sort gives the effect of a base b key with fewer than the $b + 1$ files required by an amphisbaenic sort. An example, shown in Appendix 1, is identical in structure to the example in Iverson [2 p. 191] but differs in key values because it assumes backward reading of files.

A Generalized Block Sort

In the above sorts, a pass of any block classifies the keys into b subblocks by, as far as possible, dividing the range of the keys in the block into b equal parts. We may take this as the prime principle to design a sort of keys in any range. A PL/1 program has been written to design a sort on this principle. It has as input data the key range for the file and the total number of working files. The files and keys are identified by 0-origin integers. File 0 is the input and output file. If the classification of a block not on file 0 produces at least one subblock containing only one key, then one such subblock is written to file 0. This re-

TABLE I

(1) Files $b + 1$	(2) Keys r	(3) File Reversals	(4) Passes p	(5) Theory Passes	(6) Power	(7) Revised Power
3	10	15	4.00	3.82	1.78	1.97
	100	149	7.22	7.16	1.89	1.99
	1000	1661	10.64	10.45	1.91	2.00
	10000	15461	13.91	13.80	1.94	1.99
4	10	11	2.80	2.77	2.28	2.84
	100	119	4.98	4.87	2.52	2.86
	1000	1181	7.09	6.95	2.65	2.87
	10000	11639	9.18	9.05	2.72	2.89
5	10	11	2.60	2.41	2.43	3.16
	100	109	4.24	4.07	2.95	3.40
	1000	1137	5.80	5.74	3.28	3.97
	10000	12185	7.67	7.40	3.32	3.71
6	10	11	2.50	2.23	2.51	3.16
	100	111	3.80	3.66	3.35	4.62
	1000	1051	5.27	5.10	3.70	4.26
	10000	11301	6.74	6.52	3.91	4.62
10	100	101	3.10	3.00	4.40	6.9
	1000	1019	4.20	4.03	5.16	7.0
	10000	10163	5.28	5.07	5.68	7.1
20	100	101	2.81	2.51	5.14	10.0
	1000	1039	3.66	3.30	6.58	10.0
	10000	10039	4.28	4.08	8.55	12.6

duces superfluous movement of subblocks. Appendixes 1 and 2 show typical output. The program designs an amphisbaenic sort when r is a power of b.

Table I summarizes results obtained by running the program for various values of r and b. Columns 1–4 are taken directly from the program output. Column 5 tabulates $\log_b (r) + (b - 1)/b$. This function represents an ideal minimum number of passes, since the first term is the minimum number of passes required to classify the keys, as defined by information theory, and the second term is the minimum number of passes to merge the subblocks onto the output file. A comparison of columns 4 and 5 gives a measure of the efficiency of the sort. Column 6 is the power of the sort $r^{**}(1/p)$ where p is the number of passes in column 4.

Tables are available [5, p. 76] showing the power of various merge sorting techniques. The best merge sort for six files or less is the polyphase sort, which is about equal to the block sort with $r = 1000$. With more than six files the cascade sort is usually better than the block sort for practical values of r.

Using a Block Sort

In a conventional merge sort the first pass of the data is a succession of internal sorts to produce strings of sorted keys. In succeeding passes the strings are repeatedly merged to produce a single string on the output file. When using a block sort the internal sort is performed last. The subblocks produced by the block sort must be small enough for an internal sort. Within this restriction they should, for maximum efficiency, be as large as possible. For files of nearly uniform density over the range of the key, the size of the subblocks is almost constant and can be chosen appropriately.

A variation is to perform the internal sort during the last pass to the output file. This last pass is now combined with the internal sort pass and may be ignored when calculating the power of the block sort. The revised power is shown in column 7 of Table I. For six or fewer files it is higher than any of the merge sorts, and this must make the block sort competitive in some situations. A disadvantage is that the coding for the internal sort and the block sort must be in core simultaneously, or repeatedly overlaid.

A modification of the technique will handle files whose density is not uniform. Repeated b-way block sorts are performed on the data, and a count is kept of the keys in each subblock. Whenever a sufficiently small subblock is produced, it is internally sorted and written to the output file.

The block sort can make good use of direct access storage devices. During a block sort, all the keys from a block are written to a DASD in sequence. For each subblock, we keep in core the address of the last member to be written and a count of the members. Each member as it is classified and added to the file contains the address of the previous member of its subblock. This reverse chaining will permit each subblock to be read independently for further classification or internal sorting. It is possible that by using a large number of subblocks per classification, many sorts would be completed after one classification pass. However, too many subblocks per classification increases the total seek time of the read-write head. The optimum number will depend on the file to be sorted and on the physical characteristics of the storage device.

In comparing the block sort with merge sorts generally, it will be noticed that whereas a merge sort uses a replacement type internal sort which can generate an average string length of nearly twice the storage area, in a block sort the blocks must be no larger than the storage area. This disadvantage to the block sort is more important for small files of data. In addition, a merge sort will preserve a sequence of any large portions of the file that might already be in sequence, whereas a column sort would break these existing sequences down. However, a column sort is better for sorting a file that has already been sorted on a minor key, providing this is known in advance by the program, because the minor key sequences are preserved while further classifying the data on a major key.

Acknowledgments. I am grateful to the referees for their comments and suggestions.

RECEIVED JANUARY, 1967; REVISED DECEMBER, 1967

REFERENCES

1. ASHENHURST, R. L. Sorting and arranging. Theory of switching, Rep. No. BL-7. Harvard Comput. Lab., Sect. 1, 1953.
2. IVERSON, K. E. *A Programming Language*. Wiley, New York, 1962.
3. NAGLER, H. Amphisbaenic sorting. *J. ACM 6* (Oct. 1959), 459–468.
4. PL/1: Language specifications. IBM Form No. C28-6571.
5. Sorting techniques. IBM Form No. C20-1639.

APPENDIX 1. Ashenhurst Sort

```
NUMBER OF FILES      4
NUMBER OF KEYS      10
KEYS   0 TO  9 FROM FILE  0    0 TO  2 TO FILE  1
                               3 TO  5 TO FILE  2
                               6 TO  9 TO FILE  3
KEYS   0 TO  2 FROM FILE  1    0       TO FILE  0
                               1       TO FILE  2
                               2       TO FILE  3
KEY    1       FROM FILE  2            TO FILE  0
KEY    2       FROM FILE  3            TO FILE  0
KEYS   3 TO  5 FROM FILE  2    3       TO FILE  0
                               4       TO FILE  1
                               5       TO FILE  3
KEY    4       FROM FILE  1            TO FILE  0
KEY    5       FROM FILE  3            TO FILE  0
KEYS   6 TO  9 FROM FILE  3    6       TO FILE  0
                               7       TO FILE  1
                               8 TO  9 TO FILE  2
KEY    7       FROM FILE  1            TO FILE  0
KEYS   8 TO  9 FROM FILE  2    8       TO FILE  0
                               9       TO FILE  1
KEY    9       FROM FILE  1            TO FILE  0
NUMBER OF PASSES     2.80
FILE REVERSALS      11
```

APPENDIX 2. Amphisbaenic Sort

```
NUMBER OF FILES      5
NUMBER OF KEYS      16

KEYS   0 TO  15 FROM FILE  0    0 TO  3 TO FILE  1
                                4 TO  7 TO FILE  2
                                8 TO 11 TO FILE  3
                               12 TO 15 TO FILE  4

KEYS   0 TO   3 FROM FILE  1    0       TO FILE  0
                                1       TO FILE  2
                                2       TO FILE  3
                                3       TO FILE  4

KEY    1        FROM FILE  2            TO FILE  0

KEY    2        FROM FILE  3            TO FILE  0

KEY    3        FROM FILE  4            TO FILE  0

KEYS   4 TO   7 FROM FILE  2    4       TO FILE  0
                                5       TO FILE  1
                                6       TO FILE  3
                                7       TO FILE  4

KEY    5        FROM FILE  1            TO FILE  0

KEY    6        FROM FILE  3            TO FILE  0

KEY    7        FROM FILE  4            TO FILE  0

KEYS   8 TO  11 FROM FILE  3    8       TO FILE  0
                                9       TO FILE  1
                               10       TO FILE  2
                               11       TO FILE  4

KEY    9        FROM FILE  1            TO FILE  0

KEY   10        FROM FILE  2            TO FILE  0

KEY   11        FROM FILE  4            TO FILE  0

KEYS  12 TO  15 FROM FILE  4   12       TO FILE  0
                               13       TO FILE  1
                               14       TO FILE  2
                               15       TO FILE  3

KEY   13        FROM FILE  1            TO FILE  0

KEY   14        FROM FILE  2            TO FILE  0

KEY   15        FROM FILE  3            TO FILE  0

NUMBER OF PASSES 2.75
FILE REVERSALS 17
```

Computer Construction of Project Networks

A. C. Fisher, J. S. Liebman, and G. L. Nemhauser
The Johns Hopkins University, Baltimore, Maryland*

Project networks are used in PERT and CPM. An algorithm is given for constructing project networks directly from the project precedence relations. The algorithm creates "dummy" activities and topologically orders the arcs and nodes. The number of nodes created is minimal for the given precedence relations. It has been experimentally programmed in FORTRAN II for the IBM 7094.

KEY WORDS AND PHRASES: project networks, PERT, CPM, topological ordering, network construction by computer
CR CATEGORIES: 5.32

Introduction

Networks are used in PERT (Program Evaluation Review Technique) and CPM (Critical Path Method) to represent precedence relations among various activities in a project.[1] The directed arcs of the network correspond to the activities of the project and the nodes to points at which one or more activities are completed and others are begun. The network representation of a project satisfies the property that: there is a path in the network with the arc representation $(a, \cdots, b)$ if and only if activity a must be completed before activity b can be begun. The network does not contain loops since, if activity a precedes activity b, then activity b cannot precede activity a.

An example of typical precedence relations for a very small project is shown in Table I. The corresponding project network is given in Figure 1. An algebraic representation of the network in the form of an incidence matrix is given in Table II.

TABLE I. Precedence Relations

Activity	*Immediate Successors*
a	c, d
b	d
c	—
d	—

The rows of the incidence matrix correspond to the nodes and the columns to the arcs. The entries in the matrix are 0, $\pm$ 1; +1 implies a directed arc from the node, −1 a directed arc to the node, and 0 no connection between the arc and node.

* Department of Operations Research and Industrial Engineering.
[1] Our discussion of CPM and PERT will, of necessity, be very brief. A survey with many references is given in [8].