Information Processing & Management Vol. 16, pp. 49–56
Pergamon Press Ltd., 1980. Printed in Great Britain

SORTING OF TEXTUAL DATA BASES: A VARIETY GENERATION APPROACH TO DISTRIBUTION SORTING

DAVID COOPER, MARY E. DICKER and MICHAEL F. LYNCH
Postgraduate School of Librarianship and Information Science, University of Sheffield, Sheffield S10 2TN, England

(*Received for publication* 10 *September* 1979)

Abstract—A method of sorting large textual data-bases by computer using external storage is proposed. The range of sort-keys in a sample of data to be sorted is divided into a fixed set of partitions, which should also give an adequate representation of new data from a similar source. The partitions are composed of ordered key ranges. An incoming data stream is distributed into a series of bins according to the partition in which the key lies, and the bins are then seperately sorted, using an internal sort, to give an ordered file. It is shown how the number of disc accesses needed depends on the manner in which the bins become filled, and thus on statistics of the data. Experiments using an INSPEC data-base give information on which estimates of the efficiency of the method can be based.

1. INTRODUCTION

Sorting large files of data derived from bibliographic or other primarily textual data bases is acknowledged to be an expensive procedure. The data may be word tokens from the texts of documents, author names, subject headings, etc., together with a reference to their source, and they are to be sorted in order to compare and count them (thus producing a dictionary of word types), to provide an inverted file for search purposes, to arrange them for publication in an index, or for some other similar purpose.

Because of the many transfers of data needed during the process, within internal computer storage but more significantly between internal and external stores, sorting of large files is a very time-consuming operation, and is a significant component in the cost of file inversion, index production, and other processes. Any increase in the efficiency with which sorts are performed can contribute to reduction in the costs to users of information services.

The literature on sorting procedures is extensive, and is admirably reviewed by MARTIN[1] and KNUTH[2]. Two distinctions are generally made. The first is between internal and external sorting; the former involves processing the data in internal storage, while the latter uses a combination of internal and external stores such as discs. In the data-base context, the number of items to be sorted is generally so large that external sorting is of primary importance. The second distinction is between comparison and distribution sorting methods. The first order a list of items by means of a series of comparisons of the relative magnitude of the sort keys; a wide variety of methods is available, including selection, exchanging and insertion sorting, coupled with subsequent merging from external media. Distribution sorting operates by testing a key (or part of it) against predetermined standards, and collecting together all members of a group defined by a key-range. It is normally employed only when the distribution of the keys is regular and constant. In this case, the records (which often consist of a key and a pointer) are moved into a number of bins—which may be either in internal or in external storage—each of which corresponds to a particular key range. The initial groups may then be split into smaller and smaller groups until the list is ordered, or a comparison sort may be performed on the groups themselves. The efficiency of the method depends on defining ranges to ensure an even dispersion of the keys, so it is important to know the range of values and the distribution of values of the keys in the list to be sorted.

The distribution of the initial letters of words, index terms, etc., over the alphabet is highly variable. BOURNE[3] has summarised several studies in this area. Hence the use of the initial

letters of these entities for distribution sorting is of limited value. Even more so is the use of the initial digrams (character pairs) or trigrams, since these show approximately hyperbolic rank-frequency distributions[4].

Recent research on variety generation[5], a method for mapping elements showing skew distributions onto approximately rectangular distributions, suggests an approach to sorting which may have advantages in comparison with conventional methods. The method evisaged involves the prior analysis of samples of the data to establish key-ranges between which the keys will be evenly distributed. The key-ranges form a partition set, and are represented by words at the boundaries of the partitions. In this method, each item in the incoming data stream is placed in one of a number of bins of equal size within a buffer in internal storage by comparison with the partition set. (The bins may conveniently number some power of 2.) When any bin is filled, the contents of all bins are transferred from the internal buffer to equivalent but larger storage areas on a direct access medium for cumulation. The process is continued until the whole of the file to be sorted has been processed. The contents of each partition on the external storage medium are then sorted internally to produce an ordered file.

As an illustration of the method, partition sets of sizes 4, 8, 12 and 16 were generated for the 86 words of the first paragraph of this paper, and Fig. 1 shows the ranges of words allocated to each partition in each set, together with the number of words (tokens) and the number of distinct words (types). Of course if these partition sets were to be used for larger files, the boundaries between the partitions would have to be defined accurately, so that, for example, it could be determined where each word between "author" and "bases" should go in the set of 8 partitions.

The method is related to a dictionary look-up technique studied by WALKER and GOTLIEB[6], by MEHLHORN[7] and by LYNCH[8], the purpose of which is to create a balanced binary tree which preserves lexicographic order, the position of the items or words in the tree being related to

4 partitions		Types	Tokens
a	- data	13	21
derived	- names	17	21
of	- source	18	22
subject	- word	14	22

8 partitions		Types	Tokens
a	- author	7	11
bases	- data	6	10
derived	- for	8	10
from	- names	9	11
of	- procedure	6	10
producing	- sorting	11	11
source	- thus	9	11
to	- word	6	12

16 partitions		Types	Tokens
a	- an	3	6
and	- author	4	5
bases	- bibliographic	3	5
compare	- data	3	5
derived	- expensive	5	5
file	- for	3	5
from	- in	3	5
index	- names	6	6
of	- or	2	5
order	- procedure	4	5
producing	- reference	6	6
search	- source	6	6
subject	- the	4	5
their	- thus	4	5
to		1	6
together	- word	5	6

12 partitions		Types	Tokens
a	- and	4	8
are	- be	5	7
bibliographic	- data	4	6
derived	- files	7	7
for	- headings	3	6
in	- names	7	8
of	- other	4	8
primarily	- purposes	7	7
reference	- source	7	7
subject	- them	6	8
they	- to	3	8
together	- word	5	6

Fig. 1. Partition sets generated from the words of the first paragraph of this paper.

their relative frequencies. However in the binary tree case it is desirable to place high frequency items on nodes as near to the root as possible, whereas the partition set should be chosen to place high frequency terms away from the boundaries between partitions, which are analogous to the nodes near the root.

A number of factors influence the practicability of the method proposed here. It is, clearly, dependent on characteristics of the data-base and of the individual data elements. The skewness of the rank-frequency distribution of words (related to the token-type ratio, or the mean number of occurrences of the words) differs substantially for different data-elements, e.g. author names, text words and index terms. The partition set for the same data-element in different data-bases will also depend on the particular vocabulary used, and on the relative frequencies of the words or keys within it. Further, there is the question of the constancy of these characteristics, or of their reflection in partition sets, over time.

Studies of the application of variety generation methods in several contexts provide indications of the behaviour of textual data in several of these respects. They suggest in particular that in a data-base of constant provenance there are only slow changes with time of the gross characteristics, i.e. of the efficiency with which the textual data are described by means of a relatively small symbol set comprising variable-length character strings which occur with approximately equal frequencies in the data-base. Thus LYNCH, PETRIE and SNELL[9] examined symbol sets consisting of character strings from the titles of papers in the INSPEC data-base, which showed little change over a period of three years. LYNCH[10] and BRACK, COOPER and LYNCH[11] reached similar conclusions in studying the MARC data-base. In the case of non-bibliographic texts, in this instance samples from the *Standard Sample of Present-Day Edited American English for Use with Digital Computers* (The Brown Corpus)[12], YEATES[13] and EMLY[14] have shown that, in general, only slight variations in behaviour occur within or even between different genres of informative prose in English when texts are compressed using variety-generation symbol sets of size 256. The partition sets considered here are rather different in that they are concerned with dividing an alphabetically ordered file into blocks of approximately equal size rather than with fragments of text. However the overall statistical consistency of textual data suggests that similar stability will be found for the partition sets.

2. METHODS OF IMPLEMENTATION

We now discuss two alternative strategies for storing the partitioned file on disc. A disc is divided into cylinders, each being an area of the disc which can be accessed without moving the read/write heads, and each cylinder is divided into buckets, the smallest addressable areas. The concept of a bucket is to some extent a software one, since it may be possible to select one of several sizes for it, each being a multiple of the true minimum area written or read in one operation. We assume that as internal storage bins are written to disc, each of them begins in a new bucket; this restriction could be relaxed, resulting in more efficient use of disc space, more complicated processing and more disc accesses. It is also assumed that no other programs which may be running simultaneously will use any disc used by the sorting program. If they did so the advantages of requiring only small movements of the read/write heads would be lost.

The first arrangement of partitions on the disc uses fixed disc bins. Each partition is assigned a fixed area on one or more consecutive cylinders of the disc, adjacent partitions being assigned adjacent areas of disc. When the bins are written from internal storage to disc, each can be written with one transfer except at the ends of cylinders when two will usually be needed. The seek times will be very small since adjacent partitions are stored close together on the disc, and this seek time can be partially taken up by calculations of the number of words to be transferred, the disc address and so on. Hence the writing process consists of a single sweep through the disc, with some latency time on each occasion when a new cylinder is accessed. Reading a partition into internal storage for the final sort will be rapid since the partition will be contained on a series of adjacent cylinders, and the read/write head can be positioned before reading starts. A disadvantage of this system is that a fair amount of space on the disc is wasted, since the space allocation for each partition must be sufficient to accommodate the largest.

An alternative strategy is to write all the data from internal storage to disc serially; the data to be written each time an internal storage bin is filled consist of the bins arranged con-

secutively, each preceded by a pointer to the disc address (bucket and position in bucket) of the previous bin corresponding to the same partition, and a count of the number of items in that bin. Each bin could be written beginning in a new disc bucket, but this would involve waiting for a whole rotation before writing each bin, as well as wasting space at the end of buckets. It would normally be best therefore to collapse all the internal storage bins to consecutive areas and calculate all the pointers before writing the data, with one disc transfer for each cylinder accessed. The next transfer will begin at the beginning of the next free bucket, and so the read/write heads will be correctly placed without further action. When all the data have been written to the disc each partition is read into internal storage for the final sort by a single backwards sweep through the disc, with some latency time and a small seek time as each new cylinder is accessed, the bucket address having been decided by the pointer in the previously read bin. The count in that bin enables unwanted data at the end to be excluded, as well as preventing unwanted buckets from being read.

The second disc storage pattern is more efficient than the first for writing the bins to disc but slightly less so for reading them into internal storage before the final sorts. It is shown below (Section 4. Practical Considerations) that there are likely to be more transfers involved in the former process than in the latter when the sort is used on the largest amounts of data, and since the second storage pattern also uses less disc space it is the natural choice in these circumstances. The first strategy may find a place when the sorting process is split into more than one stage (see Section 4).

3. EXPERIMENTAL WORK

An algorithm has been developed for producing a partition set for use in distribution sorting of a textual data-base. This has been used to generate several partition sets, from samples of various sizes, and their behaviour when applied to the data-base has been investigated.

The data-base used was a 6-month cumulation of the INSPEC Computer and Control Abstracts for 1976, a total of 14,696 document references. The data-element selected for examination was the free index term field; from these fields, a total of 194,663 word tokens was extracted. (A word was defined as a string of alphabetic symbols, truncated if necessary to 16 characters.) These tokens represented a total of 14,919 word types (token: type ratio = 13.05). In addition, index terms were extracted from linear samples of one record in two, one in five, and one in ten from the data-base, giving files of 97334, 38633 and 18847 word tokens. The whole file will be referred to as File A, and the samples as Files B, C and D in order of decreasing size.

From each of these four files, partition sets of sizes 16, 32, 64, 128 and 256 were generated. The generation method consisted essentially of sorting the file (using a standard disc-sort package) and then dividing up the alphabetical list according to the cumulative frequency. There are problems however when words with high frequency occur near partition boundaries and some words even have frequencies higher than the desired partition size—for example, each of the two most common words in the file, "system" and "control", has a frequency higher than 1/64 of the total number of tokens. Hence, particularly for the larger partition sets, it is necessary to allocate more than one bin to single words of high frequency. This will not cause subsequent problems of lack of storage for sorting if no other words go into the same bins as these frequent words, since they will not need to be sorted further. The problem of words of moderately high frequency causing uneveness in the sizes of partitions is harder to solve. The method chosen to reduce their effect was to divide the ordered file into sections, each section consisting either of a single high frequency word or the words between two such in alphabetical order. Partitions are then allocated to each section separately. More precisely, the steps listed below are carried out. In several of these steps some numerical parameters are specified; these are the values that were used to generate the partition sets used here, but they can be altered experimentally.

I. The ideal partition size P is calculated by dividing the total frequency by the number of partitions required.

II. The file is divided into sections. Each word with frequency above $0.5\,P$ becomes a section on its own, and the other sections are the sets of words in between. (Improved results have been obtained subsequently by altering this step, putting a word in a section on its own only if its frequency is at least P, and letting words of frequency between $0.5\,P$ and P begin a new section).

III. Two adjacent sections are fused if their combined total frequency is less than 1.3 P.

IV. The number of partitions to be allocated to each section is decided on the basis of total section frequencies. This is done by setting a "maximum level" for the mean partition size in any section, calculating the number of partitions in each section needed to achieve it, and increasing the maximum if the total number of partitions is too large. Note that the most frequent words may be allocated to more than one partition, and that these partitions will not be shared with other words. The "maximum level" used during the calculations is multiplied by 0.95 for sections containing more than one word. The balance between the amounts of space for frequent and less frequent words may be changed by altering this factor.

V. The partitions for the sections containing only one word are now fixed. For other sections, the mean partition size is calculated, and the division points for the partitions are placed as close as possible to the position required. However if a rather frequent word near a division point makes this impossible, and the result would be a partition which is smaller than 0.75 times the mean size for the section or larger than 1.3 P, then the section is divided into two at this point, and the process starts again at Step III.

The most important performance measure of a partition set with a file is the ratio of the mean number of words per partition to the maximum; this is referred to as a *density*. When partitions of the whole file are considered the density measures the efficiency with which internal storage is used on average for the final sort, and if the disc is divided into fixed sections for the partitions, it gives an upper bound to the efficiency of use of the disc space. On the other hand, the mean density of the bins in internal storage over the stages when the bins are written to disc measures the proportion of the available internal storage used on average, which is inversely proportional to the number of disc accesses required to write all the data. Note that if the latter density falls much below 0.5, then it would probably be better to abandon the fixed bins and replace them by a series of linked lists, together covering just over half the available internal storage (the exact proportion depending on the relative amounts of space needed for the record and the link). The lists can then be converted into variable length bins in the remaining internal store before writing to disc. Various modifications of this scheme could be used to provide a higher effective bin density.

Also, if only a few partitions have size near the maximum for the whole file, then the effective density can be increased if the linked bin strategy is used for disc space allocation, since special measures can be taken to deal with the few very large partitions, e.g. a small scale merge using spare capacity on the disc.

Table 1 shows the density obtained when a partition set is applied to the whole of its parent file, for each size of partition set generated from each file. There are no obvious trends according to file size, but the density decreases with increasing size of the partition set, from 0.97 for size 16 to 0.76 for size 256.

In practice a partition set will have to be generated from a sample of the data-base only. Table 2 shows the densities obtained when the partition sets derived from the small file D are applied to each file. The differences from the figures in Table 1 are small except in the case of partition sets of size 256, when there is some serious degradation of the performance of the set from the one-in-ten sample D on the whole file A.

Table 3 shows the bin densities obtained when each of the files is partitioned by each of its five partition sets, in each case for two bin sizes, one fairly small and one fairly large in relation

Table 1. Densities on application of partition sets to the whole of their parent files

	File			
Size of set	D	C	B	A
16	0·97	0·98	0·96	0·97
32	0·85	0·90	0·87	0·88
64	0·82	0·83	0·80	0·81
128	0·79	0·78	0·79	0·78
256	0·75	0·76	0·76	0·75

Table 2. Densities on application of partition sets from File D to whole files

	File			
Size of set	D	C	B	A
16	0·97	0·95	0·94	0·94
32	0·85	0·86	0·87	0·87
64	0·82	0·83	0·80	0·79
128	0·79	0·77	0·78	0·74
256	0·75	0·70	0·67	0·66

to the particular file size and partition set size. It can be seen that the densities are rather small for very small bin sizes. The last column of the table shows the result of partitioning the complete file A by means of partition sets from the smallest sample D. Comparison with the results obtained with partition sets from A suggests that the bin density should not be affected by the small sample from which the partition sets were generated except a little in the case of the sets of size 256.

Similar results for a much wider range of bin sizes are tabulated by DICKER[15]. It was generally found that the density decreased gradually from the density for the whole file, as the bin size decreased from a value large enough to accommodate all the file to a value requiring the bins to be emptied about ten times. The density then decreased more rapidly as the bin size decreased further, the relationship being approximately of the form $d = m \log b + c$, where d is the density, b is the bin size and m and c are constants.

4. PRACTICAL CONSIDERATIONS

The manner in which a sort of the type described might be implemented depends essentially on two figures; the largest number N of items which the sort is designed to take, and the number K of items which will fit into internal storage after taking account of the space needed for buffers in the initial input and final output, counters, etc., and program. It is assumed that the items are of fixed length, and that a minimum storage method of internal sorting such as "Quicksort" is chosen. Then K will represent both the number of items which can be accommodated in each of the final sorts and the number which can be stored in internal storage during the partitioning.

A natural choice for the number p of partitions is N/K. However the density must be taken into account, and p should be such that $pd(p)K \geq N$, where $d(p)$ is the overall density which can be expected with p partitions. The number p may be a power of 2 to make partitioning efficient, but this is not essential. Provision will have to be made for dealing with partitions of more than K items since the method will be applied to new data, but the general stability of characteristics such as partition sets over large quantities of data suggests that this facility will not need to be used often.

Once p has been chosen, the bin size must be not more than K/p, and will normally be chosen to be the nearest integer below K/p. The internal storage bin density D can then be predicted using tabulations such as Table 3, and the number of times the bins are emptied is about N/KD. If D is small, consideration should be given to the use of linked lists for the core bins. Alternatively, the partitioning could be done in two or more stages. In the two-stage case, a number p_1 is chosen near to the square root of p, and the partition set of size p_1^2 is chosen in two stages, the first being a set of size p_1; each partition can be divided further into p_1 partitions by the same method. The data to be sorted would be partitioned first into p_1 partitions, and the process then repeated with each of these partitions. If the disc space were allocated as fixed disc bins for the first stage, then very little extra disc space would be needed above that required for a single stage partition sort, since the disc bin for the first of the p_1 partitions could be filled with the data from the second partitioning stage as soon as it had been read into internal storage and partitioned in the second stage.

The time needed for a sort on N items is, in general, of order $N \log N$. Suppose that the time for an internal sort is $kN \log_2 N + o(N \log N)$, where k is independent of N; here $o(N \log N)$

Table 3. Mean densities in internal storage bins; partition sets generated from the file to which they are applied, except in the last column where partition sets from D are applied to A

Size of partition set	File D		File C		File B		File A		
	Bin size	Mean density	Bin size	Mean density	Bin size	Mean density	Bin size	Mean density (A)	Mean density (D)
16	500	0·94	1000	0·95	2500	0·95	5000	0·96	0·92
	100	0·79	100	0·76	500	0·85	500	0·80	0·78
32	250	0·83	500	0·86	1000	0·84	2500	0·84	0·85
	50	0·62	100	0·66	400	0·75	500	0·72	0·73
64	125	0·80	300	0·78	750	0·79	1000	0·75	0·74
	30	0·50	50	0·53	200	0·60	200	0·50	0·50
128	80	0·67	150	0·66	400	0·74	800	0·74	0·73
	20	0·39	25	0·36	100	0·48	200	0·48	0·52
256	40	0·57	80	0·62	200	0·67	400	0·68	0·61
	10	0·21	20	0·27	50	0·36	100	0·40	0·36

denotes any function f of N such that $f(N)/N \log N$ tends to zero as N tends to infinity[16], i.e. any function which is vanishingly small compared with $N \log N$ for large N. If, for some fixed p, the N items are partitioned into p equally sized bins, to be sorted separately, then each requires a time $(kN/p) \log_2 (N/p) + o(N \log N)$ for sorting, so the total sorting time is $kN \log_2 N - kN \log_2 p + o(N \log N)$, which is $kN \log_2 N + o(N \log N)$ since $kN \log_2 p$ is $o(N \log N)$. The time needed for the partitioning is of order $N \log p$, which is $o(N \log N)$. Hence a sort based on this principle will take time $kN \log_2 N + o(N \log N)$; in other words it takes asymptotically the same time as an internal sort using p times as much storage. In practice the relative efficiency of this method and external merging will depend heavily on the number of transfers to and from external storage in each method, and on the ways in which these transfers are organised. A study of these factors is now being undertaken.

5. CONCLUSIONS

The experimental results suggest that the proposed method of sorting is feasible, and that it is possible to generate a partition set from a reasonably small sample of the data to be sorted, though care must be taken with large partition sets here. The density which can be achieved in the internal storage bins seems to vary approximately linearly with the logarithm of the bin size, although further experiments are desirable to confirm this and show the closeness of the relationship.

Acknowledgements – We should like to thank the British Library R&D Dept. for providing funds to support this work, the Institution of Electrical Engineers for providing us with a data-base and JOHN MAHONEY and ALICE MCLURE for help with extracting words from the data-base.

REFERENCES

[1] W. A. MARTIN, Sorting. *Comp. Surveys* 1971, **3**(4), 147–174.
[2] D. E. KNUTH, *The art of computer programming.* Vol. 3, *Sorting and Searching.* Addison-Wesley, Reading, Mass. (1973).
[3] C. P. BOURNE, *Methods of Information Handling.* Wiley, New York (1963).
[4] R. A. FAIRTHORNE, Emperical hyperbolic distributions (Bradford-Zipf-Mandelbrot) for bibliometric description and prediction. *J. Docum.* 1969, **25**(4), 319–343.
[5] M. F. LYNCH, Variety generation—a reinterpretation of Shannon's mathematical theory of communication, and its implications for information science. *J.ASIS* 1977, **28**(1), 19–25.
[6] W. A. WALKER and C. C.GOTLIEB, A top-down algorithm for constructing nearly optimal lexicographic trees. In *Graph Theory and Computing* (Ed. by R. C. READ), pp. 303–323. Academic Press, New York, (1972).
[7] K. MEHLHORN, A best possible bound for the weighted path length of binary search trees. *SIAM J. Comp.* 1977, **6**(2), 235–239.
[8] M. F. LYNCH, The statistical microstructure of textual data-bases. *Classifications Soc. Bull.* 1978, **4**(2), 2–10.
[9] M. F. LYNCH, J. H. PETRIE and M. J. SNELL, Analysis of the microstructure of titles in the INSPEC data-base. *Inform. Stor. Retr.* 1973, **9**(6), 331–337.
[10] M. F. LYNCH, Creation of bibliographic search codes for hash-addressing using the variety-generator method. *Program* 1975, **9**(2), 46–55.
[11] E. V. BRACK, D. COOPER and M. F. LYNCH, The stability of symbol sets produced by variety generation from bibliographic data. *Program* 1978, **12**(2), 64–77.
[12] W. N. FRANCIS, *Manual of Information to Accompany 'A Standard Sample of Present-day Edited American English, for Use With Digital Computers'.* Brown University, Providence, Rhode Island, 1964.
[13] A. R. YEATES, Test compression in the Brown Corpus using variety-generated keysets, with a review of the literature of computers in Shakespearean studies. M.A. dissertation. University of Sheffield, 1977.
[14] M. A. EMLY, Compression of continuous text by n-gram encoding, with a review of the literature on computer-produced concordances to Medieval works. M.A. dissertation, University of Sheffield, 1978.
[15] M. E. DICKER, An investigation into the stability of a partitioning system applied to the INSPEC data base, and the results of a literature search on evaluation of sorting procedures. M.Sc. dissertation. University of Sheffield, 1978.
[16] G. H. HARDY, *A Course of Pure Mathematics*, 10th Edn. University Press, Cambridge, London (1952).