Atti Accademia Nazionale dei Lincei

Classe Scienze Fisiche Matematiche Naturali

Rendiconti

Rita Bernabei, Silio De Angelo, Paola Marchioro

A lexical-sorting routine

Atti della Accademia Nazionale dei Lincei. Classe di Scienze Fisiche, Matematiche e Naturali. Rendiconti, Serie 8, Vol. **58** (1975), n.3, p. 398–404.

Accademia Nazionale dei Lincei

<http://www.bdim.eu/item?id=RLINA_1975_8_58_3_398_0>

Scienza dell'informazione. — *A lexical–sorting routine.* Nota di RITA BERNABEI, SILIO D'ANGELO e PAOLA MARCHIORO, presentata (*) dal Socio G. SALVINI.

RIASSUNTO. — In questo lavoro si presenta una subroutine per ordinare alfabeticamente un elenco di $2 \div 4$ K informazioni alfanumeriche. Vengono fornite brevi istruzioni e suggerimenti per il suo uso e sono descritti gli algoritmi utilizzati.

It is well known that the problem of ordering a list of data, in compliance with a defined set of rules, does not have a unique solution and the efficiency is strongly dependent on the structure of the data, the quantity, and the ordering rules (1).

The routine we have written may be useful under the following conditions:

– a "lexical" sorting of the records is required, according to an alphanumeric ascending order, such that the hierarchy of the keys is determined by their appearance on the record, from left to right, ignoring all special characters (blanks, periods,...) embedded in them.

– all records are long and sometimes sufficiently complicated to make it unpractical to rewrite them in a form handable by other sorting routines.

– the overall volume of data is such that it can be contained in the internal memory of the computer.

A typical example may be the catalogue of a small library, consisting of a file of 1,000 records, each 20 words long, made up with surname, name and initial of the author, title of the book and its location on the shelves. In this case the ordering keys will be scattered on the record because of the different length of each element and the presence or absence of apostrophes, blanks, commas, etc. It is also clear that reconstructing 1,000 auxiliary records, in which the keys lie in fixed positions, would be a waste of time and space.

Our program is made up of a short FORTRAN routine which controls an ASSEMBLER routine for the UNIVAC, 1100 series, computers. This second subroutine has been written in Assembler language to speed up the execution time. Since it does not use any peculiar feature of these computers, it can be easily translated for any computer and, therefore it will be described in some detail.

(*) Nella seduta dell'8 marzo 1975.

(1) A systematic description and a detailed comparison of the most frequently used algorithms can be found in Ref. [1] and [2].

The calling statement of the sort routine is the following:

CALL PTSORT(MM,M,N,NORD,IFIX,IC,NC)

where:

– MM is a matrix dimensioned M×N, containing N records to be sorted, each M words long.

– NORD is an array dimensioned N, used as working area and containing, on return, the ordering of MM. This means that the content of NORD(I) indicates which record must appear in the I position of the ordering.

– IFIX is a variable containing the number of top rows of MM that one wants to " freeze " in their positions. Usually this value is set to zero. On return it will contain the number of records in which no special characters only appear. Usually the output value is equal to N.

– IC and NC are respectively the starting character and the number of characters one wants to scan in each record. If the entire record has to be considered, they have to be set equal to zero and 6×M (this value is machine dependent). As it will be shown in more detail later on, the value of IC also constrains the technique employed and therefore affects the execution time.

If IC = 0, the first character of each record must contain an effective key. In this case which, by the way, is the most frequent, the sorting is very efficient. If IC < 0, some special symbol may appear as first character. Finally, if IC > 0, this means that some extraneous information has been stored before the effective key. This situation, possible in principle, is however against the philosophy of a lexical sort and we pay for this in terms of wasted time.

The heart of the method used in this program is a bubble sort [2] accomplished using an index table contained in the array NORD. The algorithm works this way: at beginning the array NORD contains the numbers from 1 to N, in that order; then we compare the record indicated by the content of NORD(I) with the one indicated by NORD(I+1) looking for unequal keys, let them be A10 and A11. If A10 comes before A11, the ordering is right, otherwise we change the contents of NORD(I) and NORD(I+1), so that the " heavier " record " sinks " in the file while the " lighter " ones are raised toward the top of the file. The first time we repeat this comparison with I varying from 1 to N—1 and store in NVV the value of I for which the last inversion happened. Then we repeat the process just up to NORD(NVV) since the last N–NVV records are already ordered. At the end, we find, stored in NORD(1), NORD(2), ···, respectively the indices of the 1^{st}, 2^{nd}, ···, N^{th} records. The flow-chart is shown in fig. 1. Note that the α-exit of the KEYS procedure represents the return in the case that two records have all the keys equal.

In fig. 2 is shown the flow-chart of the KEYS procedure which is designed to obtain the first valid different key from the records indicated by NORD(I) and NORD(I+1). The procedure LCH and CHTEST are strongly affected by the character representation on the computer and, therefore, they will

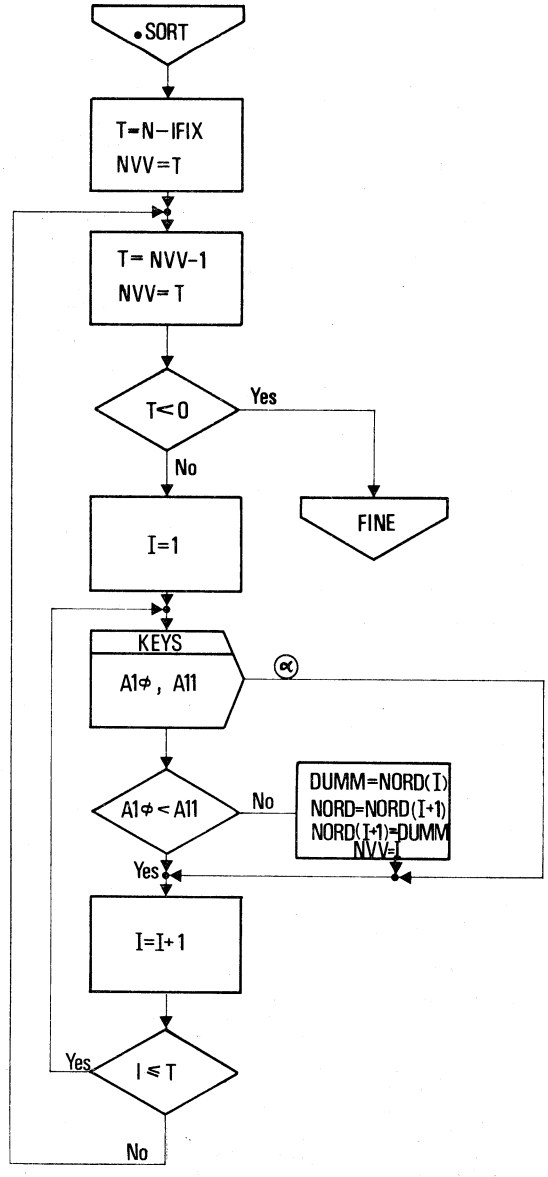

Fig. 1. – Flow–chart of the bubble sort algorithm.

not be discussed here. Their task, however, is to extract the K-th character from the record and check that is alphanumeric. Finally the CANCELL procedure places records built only of special characters at the bottom of the file and they will be ignored from now on.

The algorithm just described is fast enough, if the file is nearly ordered but very slow if it is fully disordered. Indeed it is easy to see that the execution time is roughly proportional to N in the most favorable case, and N^2 in the worst case. This happens when the last record must replace the first one. We use this method by itself, however, only if IC $>$ 0. Under these circumstances the execution time may become considerable even if the file is only 200 records long.

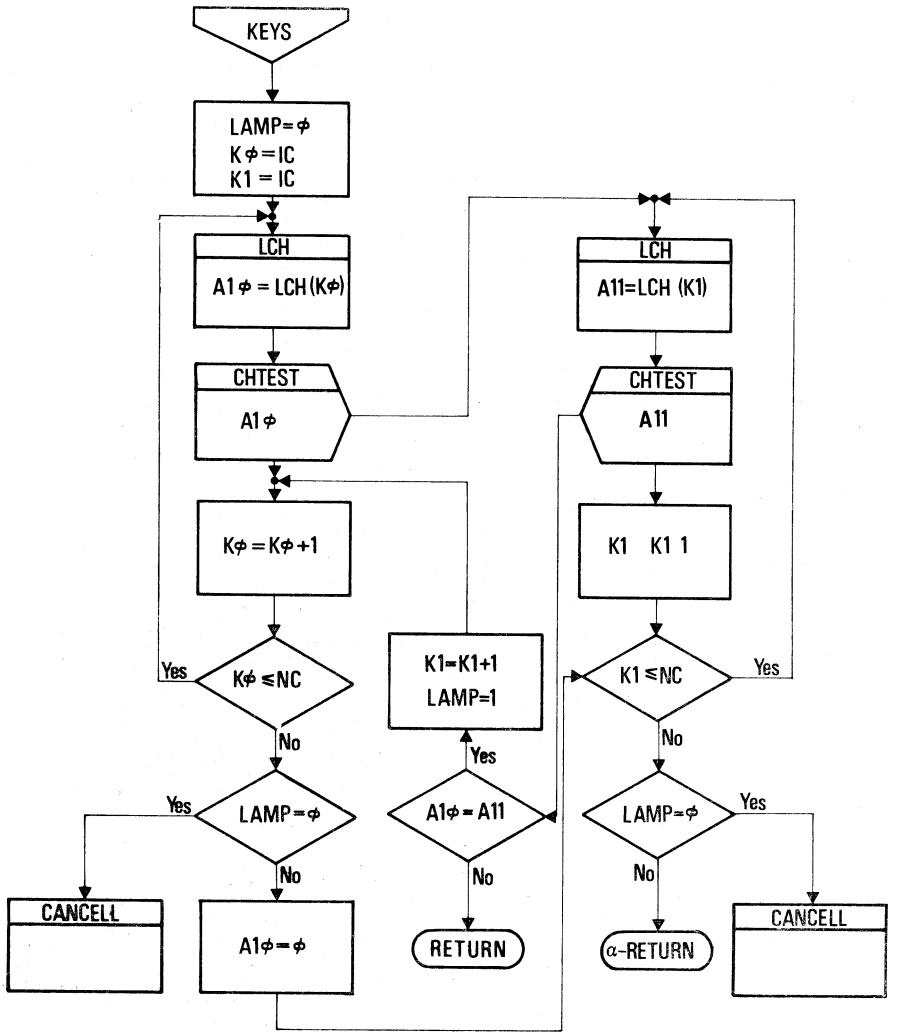

Fig. 2. – Flow-chart of the KEYS procedure.

If IC $<$ 0, we guess that the first effective key appears " usually " in the first character, but that this might be some special character " occasionally ". Under this assumption, instead of a sequence of consecutive numbers from 1 to N, we put in the array NORD a permutation of this sequence, obtained from a distribution counting sort [3], [4], examining only the

first character. Fig. 3 shows the algorithm of this techniques: CONT is an array whose components contain the number of times that the character, corresponding to that component, appears as first character on each record. To know how many records have the first character equal or less (in the sense of the ordering) than the one represented by that component, it suffices to add each component to all the preceding ones. Finally we examinate

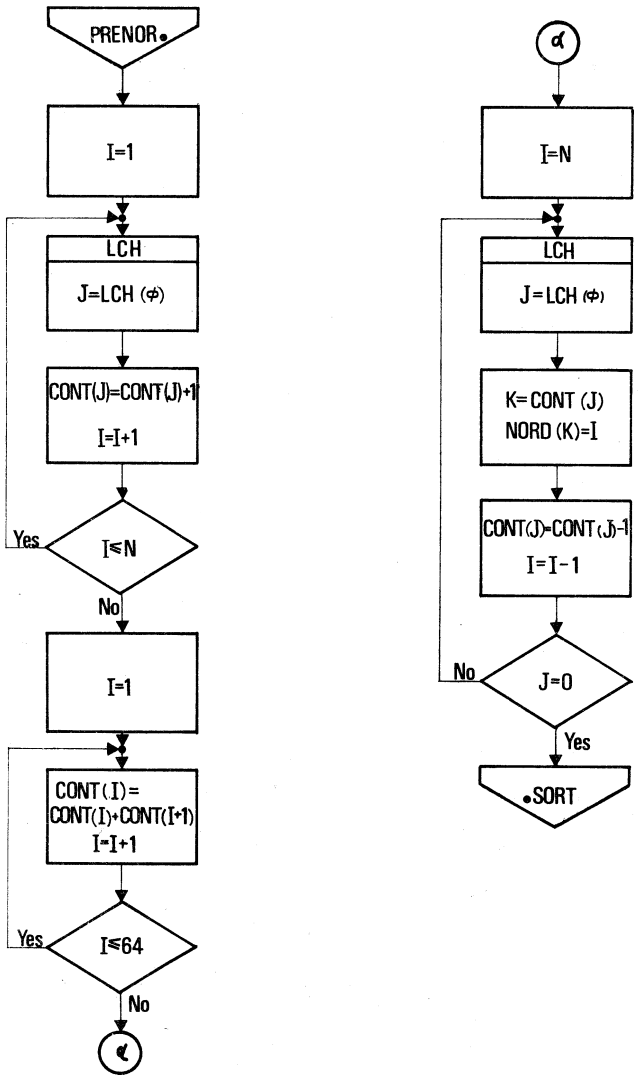

Fig. 3. – Flow–chart of the distribution counting sort.

the first character of each record and store the ordering number of that record in the component of NORD indicated by the corresponding counter. The counter is then decreased by one. In this way we obtain a partition of the file in elements of length n_i ($0 \leq n_i \leq N$, $\Sigma_i n_i = N$) such that all the records belonging to one of these elements precede or follow the records

belonging to the others, execept those corresponding to a special character. The running time is therefore reduced, in mean, to a value roughly proportional to $N \cdot \max(n_1, n_2, \cdots, n_l) \approx (1/20) N^2$.

If one can be sure that all the records begin with an alphanumeric character (IC = 0), we take advantage of this information. Then, using the distribution counting sort, this extra information guarantes that all records, belonging to a given element, precede or follow all the records belonging to another one, because the elements corresponding to special characters are empty. Then we need only sort the records within each element in turn, starting directly with the second character.

Typical running times for sorting a file of records 20 words long, are about 1/2 sec for 10 K-words, about 2 sec for 20 K and 10 sec for 40 K, on the UNIVAC 1110 computer.

In conclusion, we have presented here a routine which is particularly suitable for arranging a small library, for ordering a large bibliography, for keeping a list of experimental data divided, for example, by laboratory and/or year, and so on. In other words this program is a useful tool for all those scientific programmers which need to classify some list of alphanumeric information. In fact contrary to previous methods we emphasize that the user of this routine need not manipulate the records containing this information in order to extract the keys required by the ordering process - manipulation which is tedious to do whitin a FORTRAN program and, always, space expensive. Finally we can note that the use of a distribution counting technique, in addition to the bubble sort, allows us to maintain an acceptable running time, even for moderately large files.

APPENDIX

We want to stress the fact that the matrix MM remains unchanged. The physical ordering is, in fact, contained in the NORD vector. More precisely the numbers from 1 to N, have been permuted so that NORD(I) points to the record that has to be I^{th} in the final ordering.

If we want to rearrange the matrix MM in the proper order [5] we can use the following calling statement:

```
CALL MMORD(MM,M,N,NORD,MMAUX)
```

where:

– MM and NORD are the matrix and the array defined above, whose dimensions are M×N and N, respectively.

– MMAUX is an array, dimensioned M, used as temporary storage area, for one of the records during the re-ordering.

This routine needs only M extra locations in the main program. Fig. 4 shows the flow-chart of this routine. In this figure we have dropped the first index from the matrix MM.

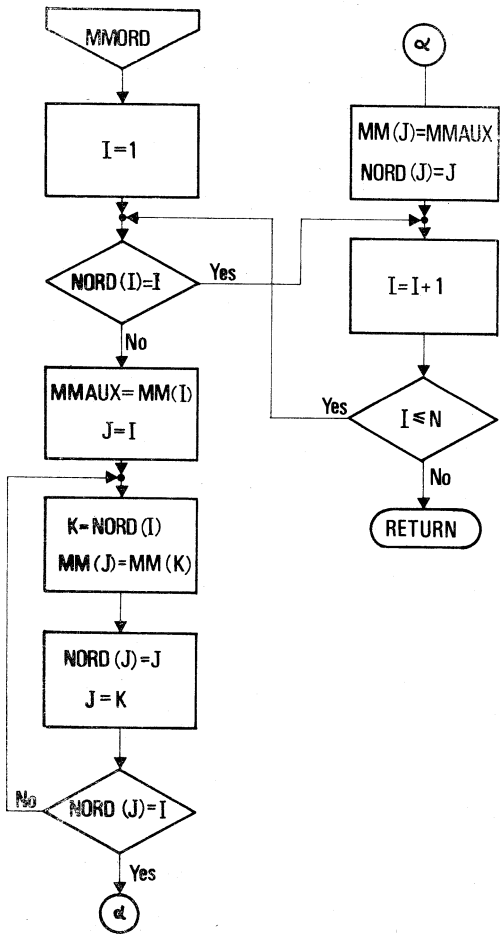

Fig. 4. – Flow-chart of the MMORD subroutine.

References

[1] Ivan Flores (1961) – *Analysis of internal computer sorting*, « J. of ACM », *8*, 41.
[2] Donald E. Knuth (1973) – *The art of Computer programming*, vol. *3*. Addison–Wesley, Reading, Massachussetts, Cap. 5.
[3] Calvin C. Gotlieb (1963) – *Sorting on Computers*, « C. of ACM », *6*, 194.
[4] Donald M. Mc Laren (1966) – *Internal sorting by radix plus sifting* « J. of ACM », *13*, 404.
[5] Donald E. Knuth (1965) – « Cybernètics », *1*, 95.