SUMMARIZING SORTS
by Robert L. McAllester

Robert L. McAllester, a Staff Analyst in Computer Usage Development Corporation's (CUDC) Palo Alto office, has been in the data processing field since 1953. He has a B.A. in mathematics from Earlham College and has also attended William Penn College, Black Mountain College and Cornell University. He is the author of "Polyphase Sorting with Overlapped Rewind", Communications of the ACM, and has taught programming courses at Heald's Business College and the College of San Mateo. He joined CUC in 1965.

Summarization is a natural by-product of sorting and should occur during all sort phases. You can define the fields to be summarized in the sort control cards just like the sort keys.

The choice of the internal sort technique can make a big difference in the amount of summarization that occurs during the internal sort phase. But summarization can result in a significant saving of overall sort time no matter what technique is chosen.

You will also realize further benefits if you expand the COBOL SORT verb definition to permit summarization.

You can include a summarizing feature in most generalized sort packages. This feature permits data to be summarized while sorting occurs. The modifications that transform a sort to a summarizing sort are usually very simple. When the comparison subroutine finds the sort keys of the two records equal, it refers the two records to the summarization subroutine. The accumulative fields of one record are added to the corresponding fields of the second. And the first is replaced by a record from the input stream just as though it had been written into the output string.

The summarization subroutine is the only additional subroutine required in the object sort program. The comparisons that are normally performed by the sort are completely adequate to define the categories to be summarized. The accumulative fields can be defined by the user in the sort control cards in the same manner as the sort keys.

The principal advantage of the summarizing sort is to shorten the intermediate and final output files that are handled by the merge passes. If any summarization occurs during the internal sort phase, the saving is accrued during all merge passes. You can estimate the amount of summarization that occurs during the internal sort phase by whatever sorting method you use.

Definitions

The following definitions are used to compare the summarizing potential of the two internal sort techniques: "Ranking by Insertion" (3) and the "Tournament Sort", (sometimes called, "Replacement-Selection" (1).)

"F" is the internal file size. The number of records that can be held simultaneously in internal storage for sorting.

"C" is the number of categories. The average number of distinct key values represented among the "F" records of the internal file.

"R" is the range of the file. The number of categories represented in the entire input file. This is the size of the summarized output file.

"L" is the gross string length. The number of input records that might be combined into each string produced by the internal sort.

"N" is the net string length. The number of summarized records expected in each string produced by the internal sort.

Tournament or Insertion

The objective of the internal sort phase is to form the input into as few strings as possible. This will minimize the number of merge passes required. This is equivalent to maximizing L. In a non-summarizing sort L=N. It is also well documented (2) that L=2F. This is true for any replacement sort technique; tournament, insertion or other.

Given the fact that the two methods will produce output strings of the same length, most programmers usually select the tournament sort over the insertion sort because the latter has a major drawback. Each time one of the F records is replaced, all records whose key values lie between the old and new records must be shifted one position to make room for the insertion and to fill in the position of the old record. Even if only the addresses of the records are shifted instead of the records themselves, this can amount to a substantial data move. This data move is not required when an incoming record is summarized with an existing record.

A summarizing sort emphasizes another important difference between these two sorting techniques. The tournament sort is an efficient method of selecting the lowest record from the set of F records, but it makes no attempt to match an incoming record to an equal record, which might already be contained in F. The summarizing insertion sort will immediately match the new record with its equal and absorb it. Consequently, in an insertion sort C=F, while in a tournament sort C≤F because of duplication of categories. If we assume a random distribution of records among the R categories, as each record is introduced to the internal sort, the chances are C among R that a member of its category is already contained among the F records of the internal sort. When a member is contained, summarization of the two records will occur immediately with an insertion sort, before output of the category in the tournament sort. The ratio of summarization during the internal sort is N/L, which is equivalent to (R-C)/R.

The Special Case R=F

The difference in the two sort methods is greatest when R=F. In this case the insertion sort will produce an in-core summary. No matter what the size of the input file, only one output string will be produced. N=R. L equals the total length of the input file.

The tournament sort will still produce the approximate results L=2F. This approximation is low in that it assumes that all the summarizations of a category will occur after the members of the next lower category have been summarized and output. The assumption is false, but the exceptions are infrequent enough to make a small difference in L.

Consequently, the output of a summarizing tournament sort will require almost the same number of merge passes as a non-summarizing sort, but the volume of each pass is restricted by the fact that no output string will exceed the value of R. At least a 50 per cent summarization will occur during the internal sort and each succeeding pass will handle only a fraction of the volume of the preceding pass. Even the less efficient tournament sort gains much power from the summarizing feature.

The summarizing tournament sort would not produce an in--core summary until $R \leq \log_2 F$. It is at this point that it becomes inevitable that each tournament series that is started by a replacement will be terminated by an equal comparison and summarization.

An Approximation Formula for Insertion Sort

The formula $L = \frac{F^2}{R-F} + 2F$ appears to approximate the expected gross string length of the summarizing insertion sort. The validity of this formula can be checked at some values. For R=F the denominator R-F=0 supporting the fact that an in--core summary occurs. As the ratio R/F becomes greater, the value of L approaches the asymptote 2F, the value of L in a non-summarizing sort.

It is possible to compute a direct comparison value for the case R=F+1. N is restricted between the limits $F < N \leq R$. Therefore, N=R. Since N/L=(R-F)/R, $L=R.^2$ The approximation formula in this case yields $L=F^2 + 2F$ or $R^2 - 1$.

Again for the case R=2F, the above logic yields the limits $2F < L \leq 4F$. The approximation formula yields L=3F which is centered between the limits.

Unequal Distribution of Categories

In the discussion above, I have assumed a random distribution in which members of each category have an equal chance of occurance. The data files to be sorted are very rarely random occurances of data. The data in these files have been very carefully organized. This organization will frequently cause members of categories to come in groups. When this occurs, the summarization of the group will occur during the internal sort phase regardless of what type of internal sort is used.

It is also common that categories to be sorted are of very unequal populations. As an example, consider a file of 100,000 records that consists of 1,000 categories. One hundred of these categories are large, accounting for half of the volume of the input file (50,000 records), to be sorted in an area F=250. The proposed approximation formula can be used to study the performance of such a file in an insertion-type, internal sort. You can assume the sort will operate like two sorts being performed simultaneously. The F record positions would be allocated into two groups, F_1 and F_2.

Since both groups of categories have an equal number of input records and the two sorts share their input and output sources, we must assume that $L_1 = L_2$. Therefore,

$$\frac{F_1^2}{(R_1 - F_1)} + 2F_1 = \frac{F_2^2}{(R_2 - F_2)} + 2F_2$$

If F=250, R_1=100, R_2=900; a root of the above equation occurs near F_1=76, F_2=174, $L_1 = L_2$=390, L=780. Then since $N = \frac{(R-F)}{R} L$, N_1=94, N_2=315 and N=409. During the internal sort, the file has been reduced to 52 per cent of its input volume.

Summarizing COBOL Sort Verb

By including the following features you could add a flexible summarizing capability to the COBOL SORT verb.

An optional EQUAL procedure would be added to the SORT statement. This would be named from the SORT statement in the same way as the INPUT and OUTPUT procedures. The EQUAL procedure is executed whenever two equal records are compared by the sort. Thus, it could be during execution of a RELEASE statement of the INPUT procedure, and during execution of a RETURN statement of the OUTPUT procedure, as well as during the merge passes.

The functions that would normally be performed in the EQUAL procedure are to add the corresponding accumulative fields of the two equal records and delete one of the records. There is no verb for deleting a record from a file; therefore, the DELETE verb must be defined for use exclusively within the EQUAL procedure.

Another problem arises from the fact that the two equal records are both members of the same file so that there is no method available for distinct identification of these two records. You can achieve distinct identification by defining a pseudo file in the DATA DIVISION. An ED file definition will supplement the SD file definition that is already required for the sort file. Then in the EQUAL procedure one record is known by the SD file name and its subordinate data names and the other by the ED file name. Assignment of the records to the two file names is completely arbitrary.

Applications of the Summarizing Sort Feature

It is often required to accumulate arrays of totals whose dimensions are unknown. These arrays may be small enough to be accumulated in memory on some occasions, but at other times require an overflow procedure. The summarizing sort verb, using an insertion type internal sort, is ideal for such applications. When the totals can be accumulated in memory, they are. When an overflow procedure is required, it is available with an efficiently preplanned merge algorithm that will summarize the totals into a single, ordered output string.

The summarizing sort verb also provides for simplification of many applications. For example, a report

containing group and subgroup totals might require that the subgroup total be shown with its percentage of the group. It is required that the group total is known before processing any of the subgroup totals. You can accomplish this by programming the INPUT procedure to release two records to the sort for each one that is released ordinarily. The second will have the subgroup identification replaced by low values. This second group of records will be summarized as a set of group totals. The first record of each group, as it is returned to the OUTPUT procedure, will be the group total record. When the group total is returned first, it can be used in page headings as well as for percentage calculations.

The practice of releasing two records for one into the sort, doubles the input volume of the sort. The resulting sort file is one of very unequal distribution among the summarizing categories. As was shown in the earlier example, such a file produces a high rate of summarization, even in the internal sort. The doubled input volume will increase the time required for the sort but will certainly not double it. The increased sort time can usually be justified by the simpler job procedures. It is not necessary to summarize the group totals separately before computing percentages.

REFERENCES

1. Goetz, M. A. "Internal and Tape Sorting Using the Replacement-Selection Technique." Comm. ACM 6 (May 1963) 201/206.

2. Gassner, B. J. "Sorting by Replacement Selecting." Comm.ACM 10 (Feb. 1967) 89-93.

3. Iverson, K. E. "A Programming Language." Wiley, 1962, p. 221.