Duplicate Record Elimination in Large Data Files

DINA BITTON and DAVID J. DeWITT
University of Wisconsin-Madison

The issue of duplicate elimination for large data files in which many occurrences of the same record may appear is addressed. A comprehensive cost analysis of the duplicate elimination operation is presented. This analysis is based on a combinatorial model developed for estimating the size of intermediate runs produced by a modified merge-sort procedure. The performance of this modified merge-sort procedure is demonstrated to be significantly superior to the standard duplicate elimination technique of sorting followed by a sequential pass to locate duplicate records. The results can also be used to provide critical input to a query optimizer in a relational database system.

Categories and Subject Descriptors: H.2.2 [**Database Management**]: Physical Design—*access methods*; H.2.4 [**Database Management**]: Systems—*query processing*

General Terms: Algorithms

Additional Key Words and Phrases: Duplicate elimination, projection operator, sorting

1. INTRODUCTION

Files of data frequently contain duplicate records which must be eliminated. Introduction of these duplicate records can occur in a number of different ways. For example, when identifying record fields (e.g., employee name) are eliminated from a data file before it is delivered to a user or an application program, a large number of duplicate records may be introduced. As another example, in relational database management systems (DBMSs) the semantics of the projection operator require that a relation be reduced to a vertical subrelation and that any duplicates introduced as a side effect be eliminated.

The "traditional" method of eliminating duplicate records in a file has been the following: first, the file is sorted using an external merge-sort in order to bring all duplicate records together; then, a sequential pass is made through the file comparing adjacent records and eliminating all but one of the duplicates. Since most operating/database systems already provide a sort facility, this approach is clearly the simplest. However, because of the expense of sorting, relational DBMSs do not always eliminate duplicates when executing a projection. Rather,

This research was partially supported by the National Science Foundation under Grant MCS78-01721, the United States Army under Contracts DAAG29-79-C-0165 and DAAG29-80-C-0041, and the Department of Energy under Contract DE-AC02-81ER10920.

Authors' address: Computer Sciences Department, University of Wisconsin-Madison, Madison, WI 53706.

the duplicates are kept and carried along in subsequent operations, thus increasing the cost of those subsequent operations. It is only after the final operation in a query is performed that the resultant relation is sorted and duplicates are eliminated.

In this paper we propose and evaluate an alternative approach for eliminating duplicate records from a file. In our approach[1] a modified external merge-sort procedure is utilized in which duplicates are eliminated as part of the sorting process. We also demonstrate that this approach has the potential of significantly reducing the execution time for eliminating duplicate records in a file.

In Section 2, algorithms for three methods of eliminating duplicates in a file are described. To evaluate the effectiveness of the modified merge-sort method, we develop, in Section 3, a combinatorial model that enables us to estimate the size of intermediate sorted runs produced by the merging process. In Section 4, we present some numerical evaluations based on this model. Our conclusions and suggestions for potential applications and extensions of our results are presented in Section 5.

2. ALGORITHMS FOR DUPLICATE ELIMINATION

Using any sorting method with the entire record taken as the comparison key brings identical records together. Since many fast sorting algorithms are known, sorting appears to be a reasonable method for eliminating duplicate records. This section briefly describes two methods for duplicate elimination which are based on sorting. The first method is an external two-way merge-sort followed by a scan that removes the duplicates. The second method is a modified version of an external two-way merge-sort, which gradually removes duplicates as they are encountered. A third technique for eliminating duplicate records utilizes a hash function and a bit array to determine whether two records are identical [2]. We will not, however, describe this method further or compare its performance with that of the other methods in Section 4 since it requires the use of specialized hardware for efficient operation.

We assume that the file resides on a mass storage device such as a magnetic disk. It consists of fixed-length[2] records that are not unique. The amount of duplication is measured by the "duplication factor," f, which indicates how many duplicates of each record appear in the file, on the average. The records are grouped into fixed-size pages. An I/O operation transfers an entire page from disk storage to the processor's memory or from memory to disk. The file spans N pages, where N can be arbitrarily large, but only a few pages can fit in the processor's memory. The cost of processing a complex operation such as sorting or duplicate elimination can be measured in terms of page I/Os because I/O activity dominates computation time for this kind of operation.[3]

[1] As part of the reviewing process, one of the referees pointed out that the sort facility used in System R [1] for executing the projection operation also utilizes this approach. This fact is, however, only documented in the listing of the sort code.

[2] We assume that records are fixed in length only to simplify the task of analyzing the relative performance of the two approaches. Our approach will work equally well whether the records are fixed or varying in length.

[3] In fact, since for algorithms such as merge-sort the sequence of pages to be read is known in advance, pages can be prefetched, enabling computation time to be completely overlapped with I/O time.

2.1 The Traditional Method

For a large data file, duplicates are usually eliminated by performing an external merge-sort and then scanning the sorted file. Identical records are clustered together by the sort operation and are, therefore, easily located and removed in a linear scan of the sorted file. We assume that the processor's memory size is about three pages and some working space. In this case, the file can be sorted using an *external* two-way merge sort [3]. First, each page is read into main memory, internally sorted, and written back to disk. Then main memory is partitioned into two input buffers and one output buffer, each large enough to hold a page. The external merge procedure starts with pairs of pages brought into memory that are then merged into sorted runs with a length of two pages. Each subsequent phase merges pairs of input runs produced by the previous phase into a sorted run twice as long as the input runs. Note that only one output buffer is required, since after one page of the output run has been produced, it can be written to disk allowing the merge operation to proceed. However, for the algorithm to be correctly executed, one must make sure either that consecutive pages of a run are written contiguously on disk, or that they are written at random locations but can be identified as consecutive pages of the same run by some address translation mechanism. With this provision made, the merge procedure can proceed and produce runs with lengths of four pages, eight pages, ..., N pages.[4] A two-way merge procedure requires $\log_2 N$ phases with N page reads and N page writes at each phase (since the entire file is read and written at each phase).

After the file has been sorted, duplicate elimination is performed by reading sorted pages one at a time and copying them in a condensed form (i.e., without duplicates) to an output buffer. Again an output buffer is written to disk only after it has been filled, except for the last buffer which may not be filled. Thus the number of page I/Os required for this stage is

$$N \text{ (reads)} + \lceil N/f \rceil \text{ (writes)}.$$

The total cost for duplicate elimination measured in terms of page I/O operations is

$$2N \log_2 N + N + \lceil N/f \rceil.$$

2.2 The Modified Merge-Sort Method

Most sorting methods can be adapted to eliminate duplicates gradually. A computational bound is established in [4] for the number of comparisons required to sort a multiset, when duplicates are discarded as they are encountered. Since we are dealing with large mass storage files, we are solely interested in working with an external sorting method. A two-way merge-sort procedure can be easily modified to perform a gradual elimination of duplicates. If two input runs are not free of duplicates, then the output run produced by merging them should retain only one copy of each record that appears in both input runs (see Figure 1). Whenever two input tuples are compared and found to be identical, only one of

[4] For the sake of simplicity, we assume that N is a power of 2. However this is not required by the algorithm. A special delimiter may be used to signal the end of a run, so that a run may be shorter than 2^i pages at phase i.

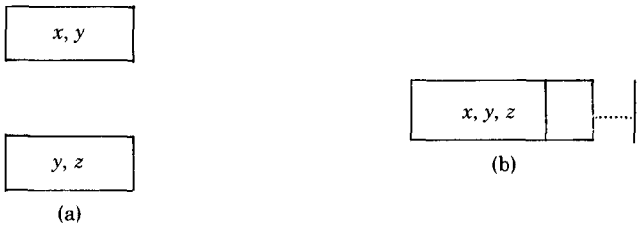

Fig. 1 Modified Merge. (a) Two input runs. (b) One output run (shorter than two input runs).

them is written to the output run and the other is discarded (by simply advancing the appropriate pointer to the next tuple). The cost of the duplicate elimination process using this modified merge-sort is then determined by two factors: the number of phases and the size of the output runs produced at each phase.

2.2.1 *Number of Phases.* The number of phases required to sort a file with graduate duplicate elimination is the same regardless of the number of duplicate tuples. That is, $\log_2 n$ merge phases are required to sort a file of n records. This is true even in the extreme case in which all the file records are identical. In this particular case, the run size is always one and every merge operation consists of collapsing a pair of identical elements into one element. By the same argument, if we start an external merge-sort with N internally sorted pages, the number of phases required is $\log_2 N$, whether or not duplicates are eliminated. However, in Section 4 we suggest a technique for reducing the number of phases required.

2.2.2 *Size of Output Runs.* Since we know the number of phases, the number of I/O operations required to execute the modified merge-sort will be completely determined if we have a method to measure how the runs grow as the modified merge-sort algorithm proceeds. When a two-way merge-sort is performed, the size of the runs grows by a factor of two at each step. However, if the merge procedure is modified in order to eliminate duplicates as they are encountered, the size of the runs does not grow according to this regular pattern. Suppose that the modified merge-sort procedure is executed without throwing away the duplicates as they are encountered. Instead, the duplicates would only be marked so that at any step of the algorithm they can be rapidly identified. Then the size of an output run produced at phase i would still be 2^i, but the number of distinct elements in the run would only be equal to the number of unmarked elements. Thus, it seems that a reasonable estimate for the average size of a run produced at the ith phase of a modified merge procedure is the expected number of distinct elements in a random subset of size 2^i of a multiset. In Section 3 we present a combinatorial model that provides us with an estimate of this value. This model is then used in Section 4 to compare the performance of the traditional method with the proposed method.

3. A COMBINATORIAL MODEL FOR DUPLICATE ELIMINATION

In this section we consider the problem of finding all the distinct elements in a multiset. A *multiset* is a set $\{x_1, x_2, \ldots, x_n\}$ of not necessarily distinct elements.

We assume that any two of these elements can be compared yielding $x_i > x_j$, $x_i = x_j$, or $x_i < x_j$. The x_i's may be real numbers or alphanumeric strings that can be compared according to the lexicographic order, or they may be records with multiple alphanumeric fields, with one (or a subset of fields) used as the comparison key. The elements in the multiset are duplicated according to some distribution $f_1, f_2, \ldots, f_m$. That is, there are f_1 elements with a "value" v_1, f_2 elements with a value $v_2, \ldots, f_m$ elements with a value v_m, and $\sum_{i=1}^{m} f_i = n$. When n is large and the values are uniformly distributed, we may assume that

$$f_1 = f_2 = \cdots = f_m = f,$$

and therefore

$$n = f * m.$$

In this case, we define f as the "duplication factor" of the multiset.

3.1 Combinations of a Multiset

Consider the following problem. Suppose we have a multiset of n elements with a duplication factor of f and m distinct elements so that $n = f*m$. Let k be any integer less than or equal to n. How many distinct combinations of k elements can be formed where all the m distinct elements appear at least once? We denote this number by $c_{fm}(k)$. We consider combinations rather than arrangements because we are interested in the identity of the elements in a subset but not in their ordering within the subset. The notation $\binom{p}{q}$ is used to represent a q-combination of p distinct elements, with the convention $\binom{p}{q} = 0$ for $q > p$.

LEMMA 1.

$$c_{fm}(k) = \binom{f*m}{k} - \binom{m}{1}\binom{f(m-1)}{k} + \binom{m}{2}\binom{f(m-2)}{k} - \cdots + (-1)^{m-1}\binom{m}{m-1}\binom{f}{k}.$$

PROOF. See Appendix

The intuitive meaning of Lemma 1 is that the number of combinations with exactly m distinct elements is equal to the number of combinations with at most m distinct elements minus the number of combinations with $m-1, m-2, \ldots,$ 1 distinct elements.

3.2 The Average Number of Distinct Elements

Starting with a multiset that has m distinct values and a duplication factor f, there are $\binom{fm}{k}$ subsets of size k. Thus, the probability that a random subset of size k contains exactly d specific distinct elements ($d \leq m$) is equal to

$$c_{fd}(k) \Big/ \binom{f*m}{k}.$$

The expected number of distinct elements in a random subset of size k can be computed by averaging over all possible values of d. The lowest possible value of d is $\lceil k/f \rceil$ since d distinct elements cannot generate a set larger than $f*d$. On the

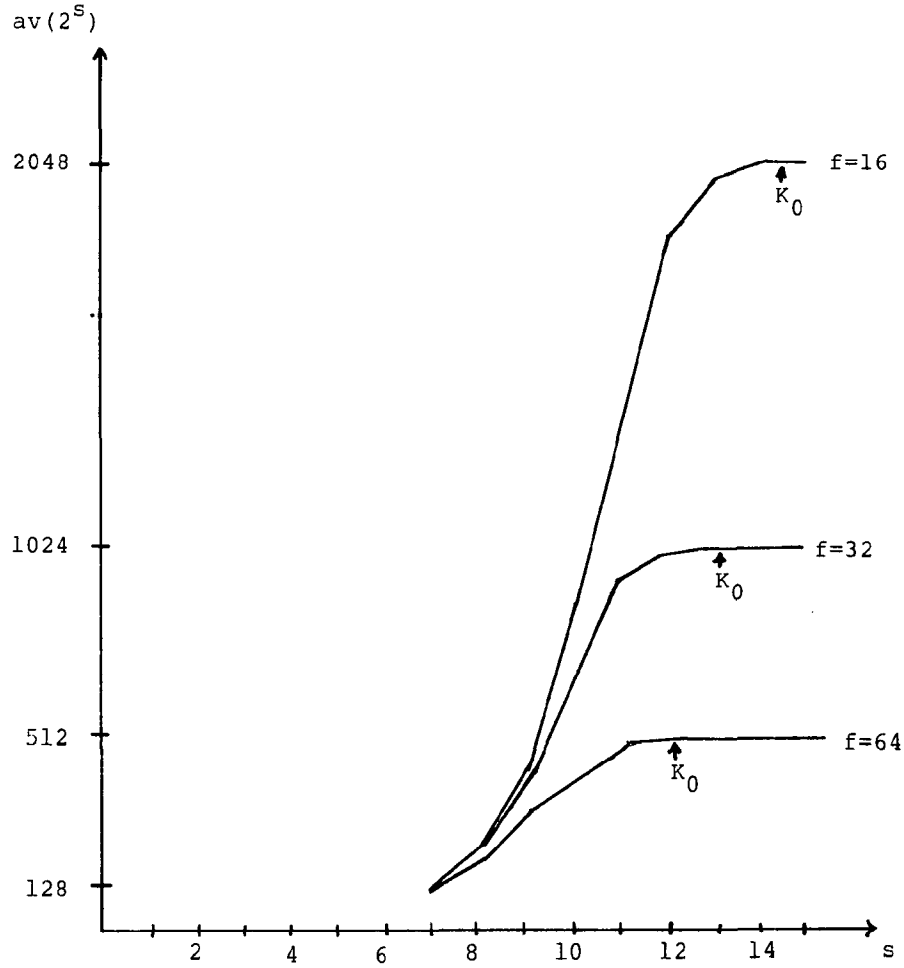

Fig. 2. Number of distinct records in successive runs.

other hand, there can be at most m distinct values since we are considering subsets of a multiset with m distinct values.

LEMMA 2. *The expected number of distinct elements in a subset of size k is*

$$av_{fm}(k) = \sum_{d=\lceil k/f \rceil}^{min(k,m)} [d*\binom{m}{d}*c_{fd}(k)] \Big/ \binom{f*m}{k}.$$

LEMMA 3. *For* $i > 1$,

$$(m - i) - (m - i + 1)*\binom{i}{1} + (m - i + 2)*\binom{i}{2} - \cdots = 0.$$

PROOF. See Appendix.

LEMMA 4. *For* $k \geq m$,

$$\sum_{d=\lceil k/f \rceil}^{k} d*\binom{m}{d}*c_{fd}(k) = m\binom{f*m}{k} - m\binom{f(m-1)}{k}.$$

PROOF. See Appendix.

THEOREM 1. *For* $k \geq m$,

$$av_{fm}(k) = m - m*\left[\binom{f(m-1)}{k} \Big/ \binom{f*m}{k}\right].$$

It is interesting to note that when f is large (i.e., the duplication factor is high), $av_{fm}(k)$ becomes a function of m and k only. This can be proven as follows:

$$\binom{f(m-1)}{k} \Big/ \binom{f*m}{k} = \prod_{i=1}^{k} \frac{f*m - f - k + i}{f*m - k + i},$$

which is approximately equal to $((m-1)/m)^k$ for large f. Therefore, $av_{fm}(k) \approx m(1 - ((m-1)/m)^k)$ for large f.

This result confirms the intuitive idea that the number of distinct elements in a random subset of a large multiset depends only on the size of the subset and on the number of distinct values in the multiset.

For smaller duplication factors, as we keep f and m constant, $av_{fm}(k)$ increases monotonically as a function of k, until for some $k = k_0$ it reaches the value m. From there it remains constant as k increases from k_0 to $f*m$. Figure 2 displays the function $av_{fm}(k)$ for $f*m = 32{,}768$ and for three different values of f (8, 16, 32). The value k_0 is of particular interest. It indicates how large a random subset of a multiset must be in order to contain at least one copy of all the distinct elements.

In the next section, we use $av_{fm}(k)$ to compare the performance of the modified merge-sort to eliminate duplicates with the performance of the "traditional" merge-sort approach.

3. COST ANALYSIS OF DUPLICATE ELIMINATION

As discussed in Section 2, the cost of a modified merge-sort is completely determined if the size of intermediate output runs can be estimated. In this section, we evaluate this cost and compare it to the cost of a traditional merge-sort. We assume that the source file consists of n nonunique records, with a duplication factor f. The modified merge algorithm will produce an output file of $m = n/f$ distinct records. Both the source file and the output file records are grouped into pages and all pages, except possibly one, contain the same number of records. The cost of duplicate elimination is measured by the number of pages read and written, assuming that the main memory can fit no more than two page input buffers and one page output buffer. When an intermediate run is produced, records are also grouped into full pages before any output buffer is written out. Only the last page of a run may not be full. Therefore, the number of pages written when an output run is produced will be:

$$\left\lceil \frac{\text{number of records in a run}}{\text{page size}} \right\rceil.$$

We assume that the external merge procedure starts with internally sorted pages, and that each of these pages is free of duplicates. This assumption is legitimate if the records are uniformly distributed across pages in the source file, and if the number of distinct records is much larger than the number of records that a single page can hold.

If there were no duplicates, the number of records in each input run read at phase i would be 2^{i-1} times the page size, since the merge procedure is started with runs that are one page long. Similarly, the number of records in an output run produced at phase i would be 2^i times the page size. Suppose the page size (measured in number of records that a page can hold) is p. Therefore, when duplicates are gradually eliminated, the expected number of records in an input run read at phase i is equal to $av_{fm}(2^{i-1}p)$, and the expected number of records in an output run produced at phase i is equal to $av_{fm}(2^ip)$, using the notation defined in Section 3. On the other hand, the number of runs produced at phase i is $n/2^ip$ (where $n = fm$ is the number of records in the source file). Therefore, the number of pages read at phase i is

$$2*\left\lceil \frac{av_{fm}(2^{i-1}p)}{p} \right\rceil * \frac{n}{2^i*p},$$

and the number of pages written is

$$\left\lceil \frac{av(2^ip)}{p} \right\rceil * \frac{n}{2^i} * p.$$

Using these formulas, we have summarized in Table I the total number of page I/Os required to eliminate duplicates from a file of 131,072 records. With 128 records per page, this file spans 1024 disk pages. We have considered various duplication factors from 2 (i.e., there are 2 copies of each record) to 64 (i.e., there are 64 copies of each record). The results indicate that a modified merge-sort requires substantially fewer page I/Os than a standard merge-sort, especially when the amount of duplication is large. When a standard merge-sort is used to eliminate duplicates, it must be followed by a linear scan of the sorted file. Therefore, we also show this augmented cost in the rightmost column of Table I.

There are, however, situations in which the cost of the linear scan can be ignored. Consider, for example, the case of a complex relational query in which a projection is immediately followed by a join. In this case duplicate tuples can be eliminated as they are presented to the join and thus there is no need for an explicit elimination step after sorting.

A further reduction of page I/Os can be achieved by terminating the modified merge procedure as soon as the runs have achieved the result file size. When this happens, all the output runs will essentially be identical and each of them will contain all the distinct records. As we observed in Section 3, this may occur a few phases before the final phase, for example, at phase number $(\log_2 N) - i$, for some $i > 1$ (N being the number of pages spanned by the source file). When this phase is reached, a single run may be taken as the result file since it contains all the distinct records and no duplicates. Therefore, the elimination process is complete and one may save the additional I/O operations, which serve only to collapse together several identical runs. Table II shows the I/O cost of this "shortened" procedure, compared to the cost of a complete modified merge sort. For this file size, the savings in page I/Os can reach up to 7 percent of the total cost. For a smaller file size (32K records) and a small duplication factor, we have observed an improvement on the order of 10 percent. When varying the file size and the duplication factor, we have observed that the improvement was greater

Table I. Cost of Duplicate Elimination

f	Modified merge	Standard merge	Standard merge + scan
2	19008	20480	22046
4	17400	20480	21760
8	15664	20480	21632
16	13840	20480	21568
32	12000	20480	21536
64	10192	20480	21520

Table II. Early Termination of Modified Merge

f	Modified merge	Shorter merge	Improvement
2	19008	17728	1280
4	17400	16264	1136
8	15664	15280	384
16	13840	13264	576
32	12000	11328	672
64	10192	9472	720

for very small or very large duplication factors, while it was smaller in the midrange values (e.g., $f = 8$ and $f = 16$ in Table II).

Since we have only estimated the expected size of the runs, our numbers are only accurate provided that the actual run size does not fall too far away from that average. This will certainly not happen if the records are uniformly distributed in the source file. Finally, it is very important to note that if there is no a priori information about the number of duplicate records present in the source file, the modified merge-sort can still be used to eliminate duplicates and the procedure can be terminated as soon as the run size stops growing. When this condition is verified, a single run can be taken as the result file, although a precise statement about the probability that such a run indeed contains *all the distinct records* requires a more elaborate statistical model than the one we have used.

5. CONCLUSIONS

A model for evaluating the cost of duplicate elimination has been presented. We have shown how, by modifying a two-way merge-sort, duplicates can be gradually eliminated from a large file at a cost which is substantially less than the cost of sorting. Accurate formulas have been established for the number of disk transfers required to eliminate duplicates from a mass storage file. These formulas can be used whenever there exists an a priori estimate for the amount of duplication present in the file. When such an estimate is not available, it is argued that the modified merge-sort method should still be used. In this case, a condition for testing that all duplicates have been removed is described.

We have based our analysis on a combinatorial model that characterizes random subsets of multisets. Only a particular category of multisets has been considered, where all elements have the same order. Thus, our results are only accurate for files with a uniform duplication factor (i.e., each record is replicated the same number of times in the entire file). Refining our analysis would require

the use of more sophisticated statistical tools to model the distribution of duplicates more accurately. However, for files with large numbers of records and with many duplicates, our model would provide a reasonable approximation.

The model developed in this paper can serve as a tool to be used by a query optimizer in estimating the cost of eliminating duplicates from a relation. Using this estimated cost an optimizer can schedule operations so that the total execution time of the query is minimized.

APPENDIX

LEMMA 1.

$$c_{fm}(k) = \binom{f*m}{k} - \binom{m}{1}\binom{f(m-1)}{k} + \binom{m}{2}\binom{f(m-2)}{k} - \cdots + (-1)^{m-1}\binom{m}{m-1}\binom{f}{k}.$$

PROOF. To prove the lemma, we express the total number of combinations of size k in terms of the number of combinations of size k with m distinct elements, the number of combinations of size k with $m-1$ distinct elements, and so forth.

$$\binom{f*m}{k} = c_{fm}(k) + \binom{m}{1}c_{f(m-1)}(k) + \binom{m}{2}c_{f(m-2)}(k) + \cdots$$

$$\binom{f(m-1)}{k} = c_{f(m-1)}(k) + \binom{(m-1)}{1}c_{f(m-2)}(k) + \binom{(m-1)}{2}c_{f(m-3)}(k) + \cdots.$$

By combining these expressions to form the right-hand side sum in Lemma 1, all the $c(k)$ cancel each other except for $c_{fm}(k)$. Notice also that k might be greater than $f(m-i)$ for some $i > 0$, which according to our notation would imply that some of the terms in the right-hand side sum become zero. □

LEMMA 3. *For* $i > 1$,

$$(m-i) - (m-i+1) * \binom{i}{1} + (m-i+2) * \binom{i}{2} - \cdots = 0.$$

PROOF. Let us consider the product $x^{m-i}(1-x)^i$. By expanding the second factor,

$$x^{m-i}(1-x)^i = x^{m-i} - \binom{i}{1}x^{m-i+1} + \binom{i}{2}x^{m-i+2} - \cdots$$

and

$$\frac{\partial}{\partial x}[x^{m-i}(1-x)^i] = (m-i)x^{m-i-1} - (m-i+1)\binom{i}{1}x^{m-i} + (m-i+2)\binom{i}{2}x^{m-i+1} - \cdots.$$

For $x = 1$ and $i > 1$ this derivative is equal to zero. □

LEMMA 4. *For* $k \geq m$,

$$\sum_{d=\lceil k/f \rceil}^{k} d * \binom{m}{d} * c_{fd}(k) = m\binom{f*m}{k} - m\binom{f(m-1)}{k}.$$

PROOF. Let

$$S = \sum_{d=\lceil k/f \rceil}^{m} d * \binom{m}{d} * c_{fd}(k).$$

Since for each k, $c_{fd}(k)$ is a linear combination of terms of the form $\binom{f(m-i)}{k}$ and since the upper bound for d in S is m, S may be rewritten backwards as a linear combination of terms of the form $\binom{f(m-j)}{k}$, j = 0, 1, ···. Then the coefficient of $\binom{fm}{k}$ is $m\binom{m}{m}$. The coefficient of $\binom{f(m-1)}{k}$ is $(m-1)\binom{m}{1} - m^2 = -m$.

For $i > 1$, the coefficient of $\binom{f(m-1)}{d}$ is

$$(m-i) - (m-i+1)\binom{i}{1} + (m-i+2)\binom{i}{2} - \cdots$$

which is null by Lemma 3. □

ACKNOWLEDGMENT

We gratefully acknowledge the comments and helpful suggestions made by Haran Boral.

REFERENCES

1. ASTRAHAN, M., BLASGEN, M.W., CHAMBERLIN, D.D., ESWARAN, K.P., GRAY, J.N., GRIFFITHS, P.P., KING, W.F., LORIE, R.A., MCJONES, P.R., MEHL, J.W., PUTZOLU, G.R., TRAIGER, I.L., WADE, B.W., AND WATSON, V. System-*R*: A relational approach to database management. *ACM Trans. Database Syst. 1*, 2 (June 1976), 97–137.
2. BABB E. Implementing a relational database by means of specialized hardware. *ACM Trans. Database Syst. 4*, 1 (March 1979), pp. 1–29.
3. KNUTH, D.E. *The Art of Computer Programming*, Vol. 3. Addison-Wesley, Reading, Mass., 1973.
4. MUNRO, I., AND SPIRA, P.M. Sorting and searching in multisets. *Siam J. Comput. 5*, 1 (March 1976).

Received October 1981; revised August 1982; accepted September 1982.