EXHIBIT 5

PART 4 OF 6

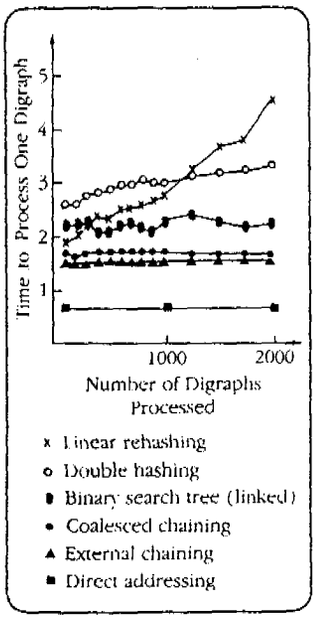


Figure 7.48
Frequency analyses of digraphs. Hashed representations. Average measured time per digraph processed.

The results shown in Figure 7.48 are those expected. Direct addressing, with one probe per digraph, gives the best performance. External chaining and coalesced chaining give essentially the same performance. Using a binary search tree gives better performance than either linear or double hashing, despite requiring more probes during a search. This is because searching a hash table requires more overhead: evaluating the hash function and the array mapping function. If the data structure were bigger, the difference in number of probes required would be larger and the hash techniques would provide better performance than would a binary search tree.

Note that initially, linear probing is faster than double hashing. This is because clusters have little effect on performance, whereas collisions require an extra computation (the step size) for double hashing. As the hash table fills and clusters form, the performance of linear probing deteriorates.

Another consideration for hashing is worst-case performance. Table 7.2 shows the maximum number of probes required to find a digraph as a function of the number of digraphs processed.

Observe that for coalesced and external chaining the maximum number of probes is little different from the average. For linear rehashing and, to a lesser extent, double hashing the maximum number of probes becomes quite large as the table fills. In the worst case, every element hashes to the same home address, and the performance can be disastrous. A general discussion of the worst-case performance of hash files, which compares linear searching, random hashing, and external chaining, is given in Larson (1982).

TABLE 7.2 Comparison of hashing techniques, including the maximum number of probes for a successful search. The table size = 500 and the cellar size for coalesced chaining is 43.

			Linear search		Double hashing		Coalesced chaining		External chaining	
Digraphs processed	**Distinct digraphs**	**Load factor**	**Avg probes**	**Max probes**	**Avg probes**	**Max probes**	**Avg probes**	**Max probes**	**Avg probes**	**Max probes**
500	165	0.55	1.35	6	1.41	7	1.15	2	1.15	2
1000	225	0.75	2.44	57	1.72	7	1.27	3	1.22	3
1500	256	0.85	5.44	105	2.12	23	1.30	3	1.25	3

7.6.2 *Frequency Analysis Summary*

Let us now review and compare all the approaches we have used for our frequency analysis problem. We have used three data structures: lists, binary trees, and sets. The analysis with lists involved two representations—array and linked—and four different orders—chronological, sorted, frequency ordered, and self-organizing. The analysis with binary trees used linked representations of both binary search trees and AVL trees. The analysis using sets was based

on an array representation of a hash table and considered four collision-resolution policies—linear rehashing, double hashing, coalesced chaining, and external chaining.

Figure 7.49 shows the expected search lengths for five representative data structures—sequential search of a list in chronological order, binary search of a sorted list, binary search tree, hashing with linear rehashing, and hashing with coalesced chaining. A semilog plot is used because of the range of search lengths.

Figure 7.50 compares the performance of the data structures considered in Figure 7.49. A semilog plot is used so that differences can be clearly seen. A comparative analysis of data structures includes performance (both average and worst case), memory requirements, algorithm complexity, and particular strengths and weaknesses. Figures 7.49 and 7.50 show the results of empirical studies of performance. Memory requirements are discussed in the chapters in which we introduced the data structures. All of the data structures in this text are (relatively) simple. We can, however, see one impact of complexity by comparing Figures 7.49 and 7.50. Sequential search of a list requires many more probes than does binary search of a sorted list. However, because one probe during a binary search is more complex than one probe during a sequential search, the number of elements must be large before a binary search pays off. The meaning of "large" varies with the processor and software used, and for our example it appears that large means 150 or more elements.

An example of a data structure weakness is the order of hashed data. Since the elements are stored in random order, it would be difficult to find all elements whose key values are in a given interval. It would be easy to do this for data in a sorted list, and that is an advantage of that data structure. As we suggest here, selection of a data structure involves matching application requirements with data structure strengths. We have seen in the case of a simple problem, frequency analysis of digraphs, that there are many trade-offs to consider.

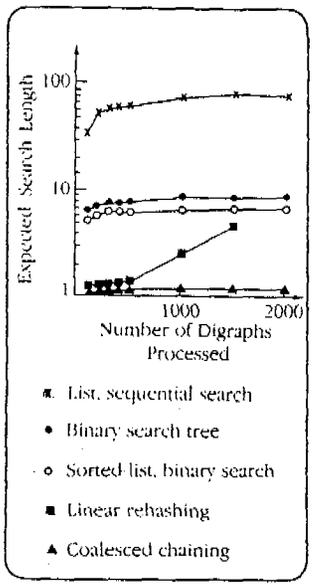


Figure 7.49
Frequency analysis of digraphs. Comparison of methods—log plot. Expected search length.

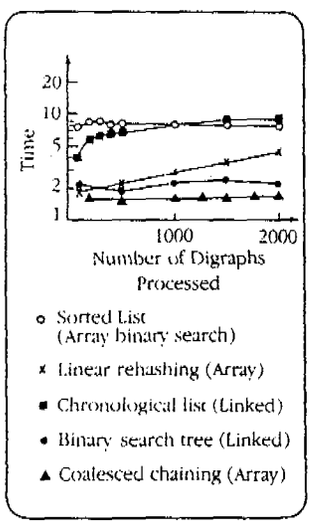


Figure 7.50
Frequency analysis of digraphs. Comparison of methods—log plot. Average measured time per digraph processed.

7.7 Summary

In this chapter we have studied data structures whose structure is setlike. The only relationship of interest among the elements is that they are members of the set. We have distinguished between two kinds of sets—those whose elements are atomic and those whose elements are structured.

We have seen that a particularly important implementation technique for sets of structured elements is hashing. We studied several hashing functions and several collision-resolution (or rehashing) strategies. The rehashing strategies upon which we concentrated were linear rehashing and double hashing (both open address methods) and external and coalesced chaining. Performance analyses and comparisons were made in Sections 7.5 and 7.6. They clearly showed that hashing is an excellent method in terms of the execution times of its algorithms.

One question that we have not yet addressed is a comparison of the

complexity of hashing algorithms with those of lists and trees. We have given no complete package for the hashed implementation of sets, but rather we have presented several of the key algorithms. In previous chapters we closed by giving tables containing at least crude measures of the complexity of the various packages presented in the chapter. As a substitute here, Table 7.3 compares the complexity of a single operation—*findkey*—for various data structures and their implementations. We see that in terms of the simple measure we are using, complexity of the hashed (open address) implementations of *findkey* are comparable with those of the other data structures.

TABLE 7.3 Complexity of *findkey* for various data structures.

Data type	Representation	Lines of code for *findkey*
List	Array	9
List	Linked	14
Sorted list	Array (binary search)	17
Binary search tree	Linked	12
Set	Hashed, linear rehash	13
Set	Hashed, double hashing	15

The applied problems section at the end of the book suggests additional applications of the data structures discussed in this chapter. Problems related to this chapter's topics are 2, 3, 4, 5, and 10.

US006571019B1

(12) **United States Patent**
Kim et al.

(10) **Patent No.: US 6,571,019 B1**
(45) **Date of Patent: May 27, 2003**

(54) **APPARATUS AND METHOD OF ENCODING/ DECODING A CODED BLOCK PATTERN**

(75) Inventors: **Jae-Kyoon Kim**, Seoul (KR); **Jin-Hak Lee**, Taejon (KR); **Kwang-Hoon Park**, Inchon (KR); **Joo-Hee Moon**, Seoul (KR); **Sung-Moon Chun**, Seoul (KR); **Jae Won Chung**, Kyoungki-do (KR)

(73) Assignee: **Hyundai Curitel, Inc** (KR)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/306,743**

(22) Filed: **May 7, 1999**

Related U.S. Application Data

(63) Continuation-in-part of application No. 08/740,300, filed on Oct. 25, 1996, now abandoned.

(30) **Foreign Application Priority Data**

Oct. 26, 1995 (KR) 95-37918
Oct. 5, 1996 (KR) 96-44049

(51) **Int. Cl.**[7] **G06K 9/36**; H04N 7/12
(52) **U.S. Cl.** **382/246**; 375/240.23
(58) **Field of Search** 382/236, 239, 382/241, 242, 243, 246, 245, 250, 251, 266, 276, 305; 375/240.03, 240.18, 240.2, 240.23, 240.24; 341/67, 65; 358/539; 348/14.13

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,220,325 A * 6/1993 Ackland et al. 341/67
5,295,201 A * 3/1994 Yokohama 382/236
5,335,016 A * 8/1994 Nakagawa 375/240.03
5,414,469 A * 5/1995 Gonzales et al. 375/240.18
5,559,557 A * 9/1996 Kato 375/240.03
5,563,593 A * 10/1996 Puri 341/67
6,192,081 B1 * 2/2001 Chiang et al. 375/240.16

* cited by examiner

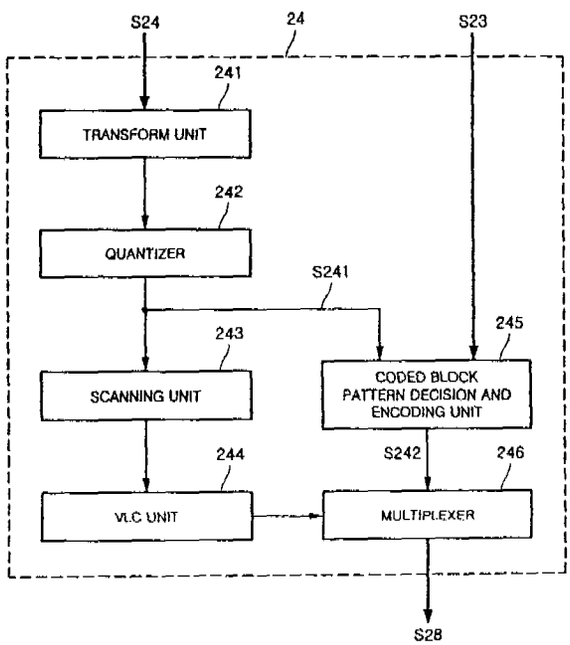
Primary Examiner—Bhavesh M . Mehta
Assistant Examiner—Kanji Patel
(74) *Attorney, Agent, or Firm*—Blakely Solokoff Taylor & Zafman

(57) **ABSTRACT**

The present invention selectively applies one of VLC tables stored in a memory for encoding a coded block pattern of a macroblock according to the number of blocks having an object within the macroblock, the number of blocks obtained using shape information, thereby reducing the amount of data transmitted and increasing coding efficiency.

The present invention also selectively applies one of VLD tables stored in a memory for decoding a coded block pattern of a macroblock according to the number of blocks having an object within the macroblock, the number of blocks obtained using shape information.

9 Claims, 14 Drawing Sheets



US005121495A

United States Patent [19]

Nemes

[11] Patent Number: 5,121,495

[45] Date of Patent: Jun. 9, 1992

[54] METHODS AND APPARATUS FOR INFORMATION STORAGE AND RETRIEVAL UTILIZING HASHING TECHNIQUES

[75] Inventor: Richard M. Nemes, Brooklyn, N.Y.

[73] Assignee: Bell Communications Research, Inc., Livingston, N.J.

[21] Appl. No.: 430,901

[22] Filed: Oct. 31, 1989

Related U.S. Application Data

[63] Continuation of Ser. No. 151,639, Feb. 2, 1988, abandoned.

[51] Int. Cl.[5] G06F 15/411; G06F 12/00

[52] U.S. Cl. 395/600; 364/222.81; 364/222.82; 364/281.1; 364/282.1; 364/252.3; 364/DIG. 1

[58] Field of Search ... 364/200 MS File, 900 MS File

[56] References Cited

U.S. PATENT DOCUMENTS

4,121,236	10/1978	Venton et al.	364/900 X
4,215,402	7/1980	Mitchell et al.	364/200
4,447,875	5/1984	Bolton et al.	364/200
4,582,113	2/1985	Hagenmaier, Jr. et al.	364/200
4,716,[illegible]	12/1987	Oxley et al.	364/200
4,775,932	10/1988	Oxley et al.	364/200

OTHER PUBLICATIONS

"The Art of Computer Programming", Sorting and Searching, D. E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 506–549, 1973.

"Data Structures with Abstract Data Types and Pascal", D. F. Stubbs and N. W. Webre, Brooks/Cole Publishing Company, 1985, Section 7.4, Hashed Implementations, pp. 310–336.

"Data Structures and Program Design", R. L. Kruse, Prentice-Hall, Inc. 1984, Section 3.7, Hashing, pp. 112–126.

Primary Examiner—Gareth D. Shaw
Assistant Examiner—Paul Kulik
Attorney, Agent, or Firm—James W. Falk

[57] ABSTRACT

A method and apparatus for performing storage and retrieval in an information storage system is disclosed which uses the hashing technique. In order to prevent contamination of the storage medium by automatically expiring records, a garbage collection technique is used which removes all expired records in the neighborhood of a probe into the data storge system. More particularly, each probe for insertion, retrieval or deletion of a record is an occasion to search the entire chain of records found for expired records and then removing them and closing the chain. This garbage collection automatically removes expired record contamination in the vicinity of the probe, thereby automatically decontaminating the storage space. Because no long term contamination can build up in the present system, it is useful for large data bases which are heavily used and which require the fast access provided by hashing.

8 Claims, 6 Drawing Sheets

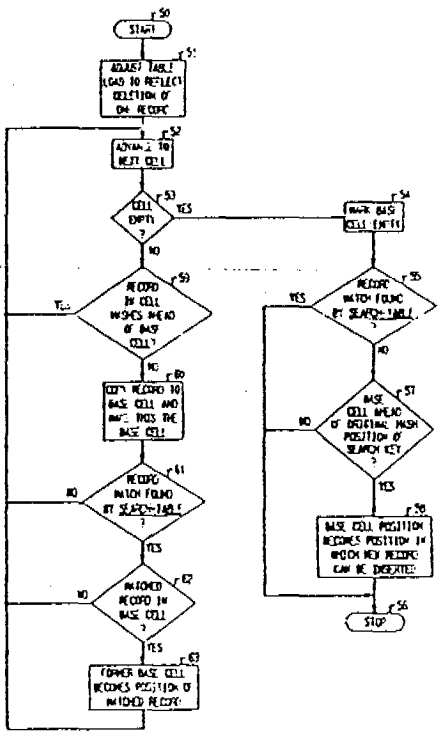


FIG. 1

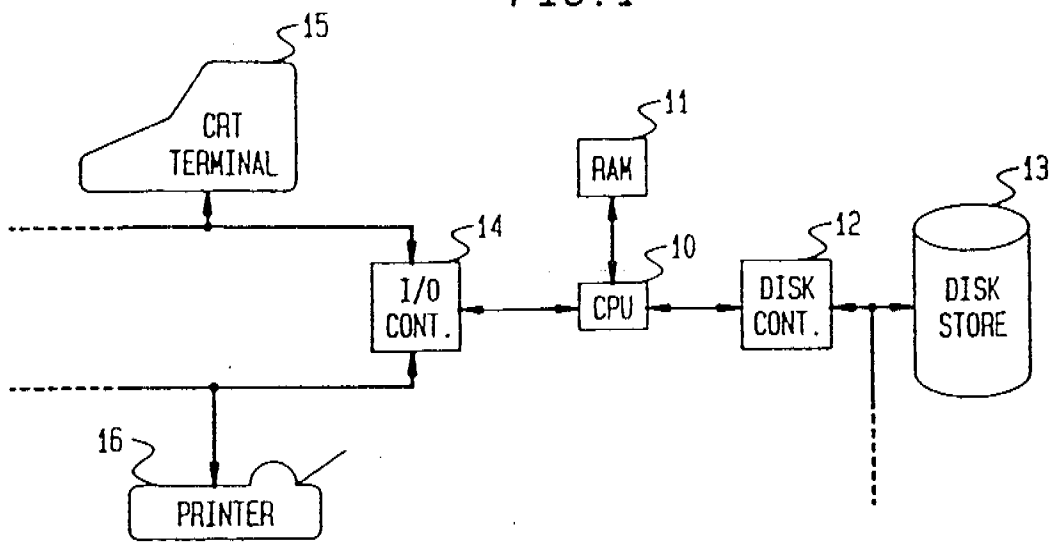


FIG. 2

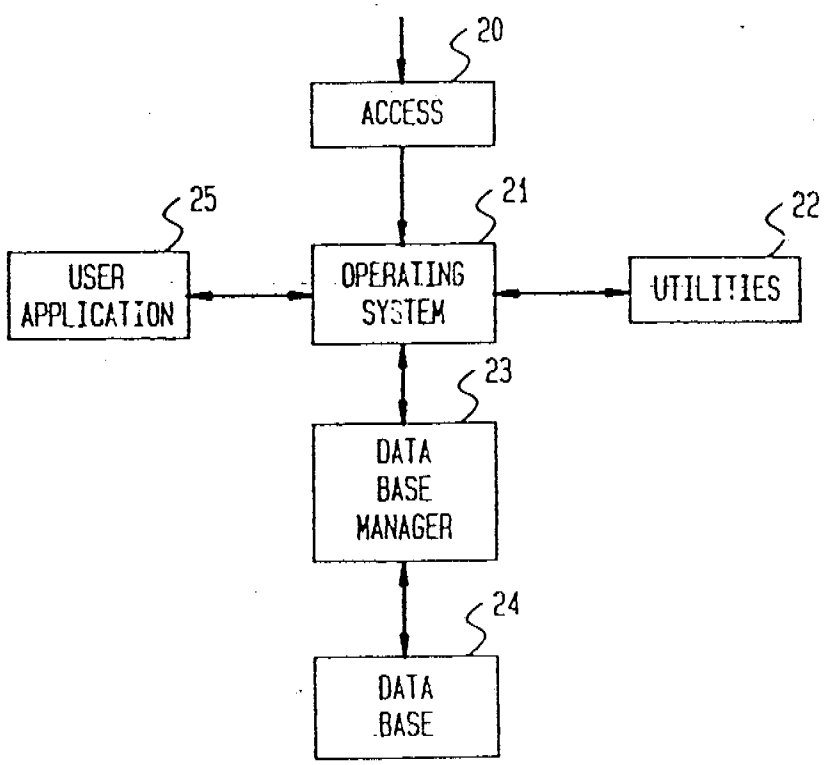


FIG. 3

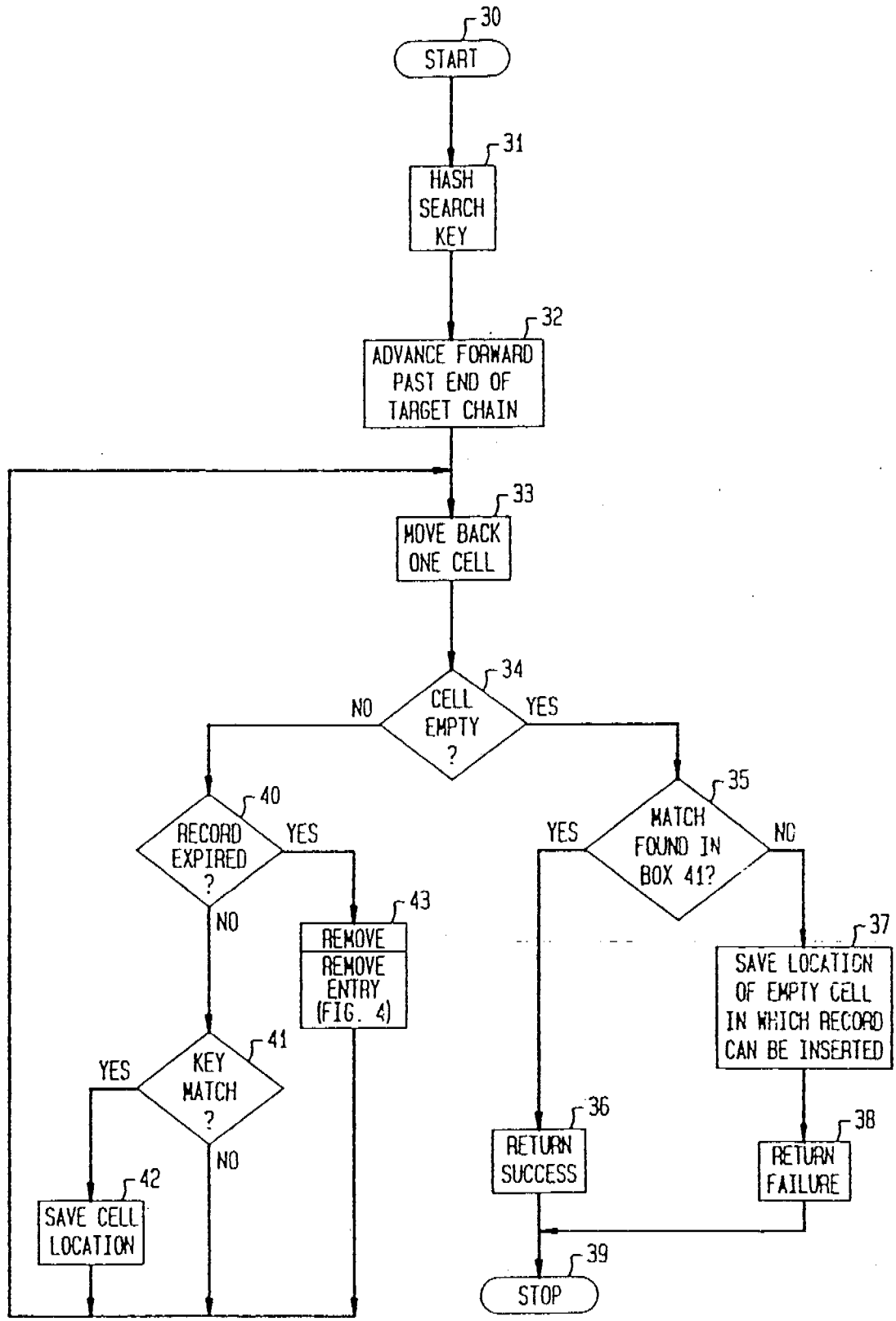

30
START
31
HASH SEARCH KEY
32
ADVANCE FORWARD PAST END OF TARGET CHAIN
33
MOVE BACK ONE CELL
34
CELL EMPTY ?
NO
YES
40
RECORD EXPIRED ?
YES
NO
43
REMOVE
REMOVE ENTRY (FIG. 4)
41
KEY MATCH ?
YES
NO
42
SAVE CELL LOCATION
35
MATCH FOUND IN BOX 41?
YES
NO
36
RETURN SUCCESS
37
SAVE LOCATION OF EMPTY CELL IN WHICH RECORD CAN BE INSERTED
38
RETURN FAILURE
39
STOP

FIG. 4

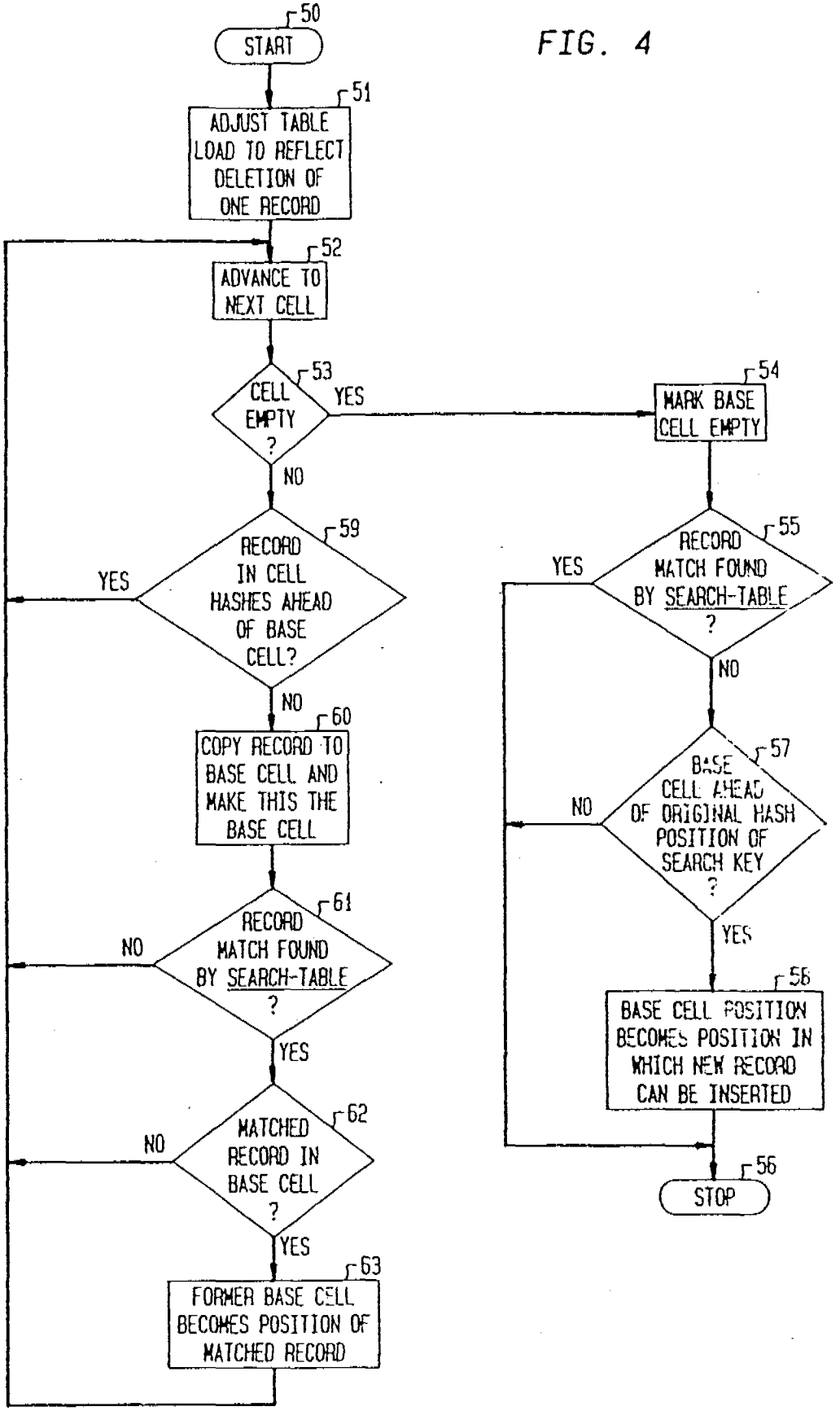


FIG. 5

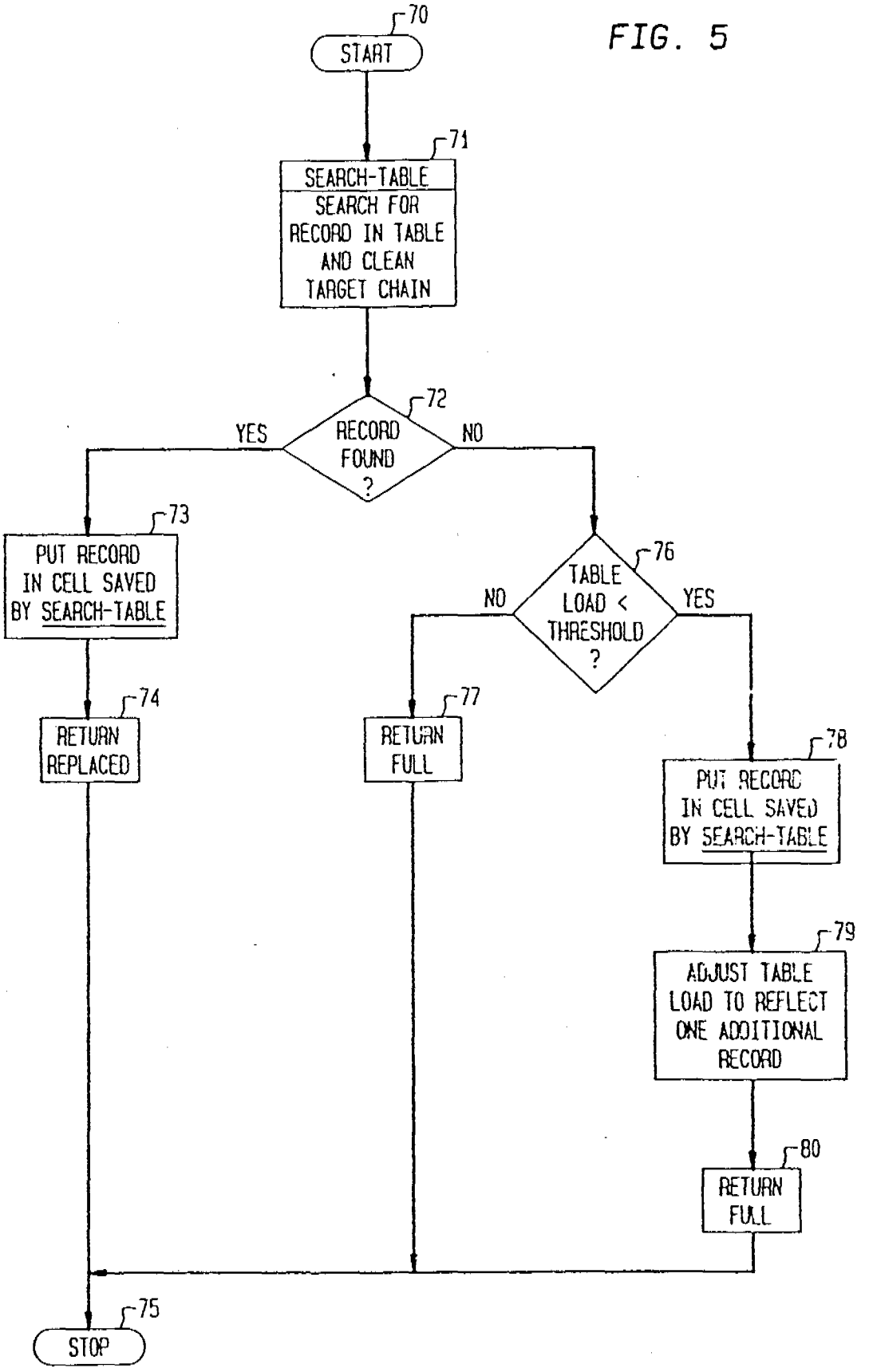


FIG. 6

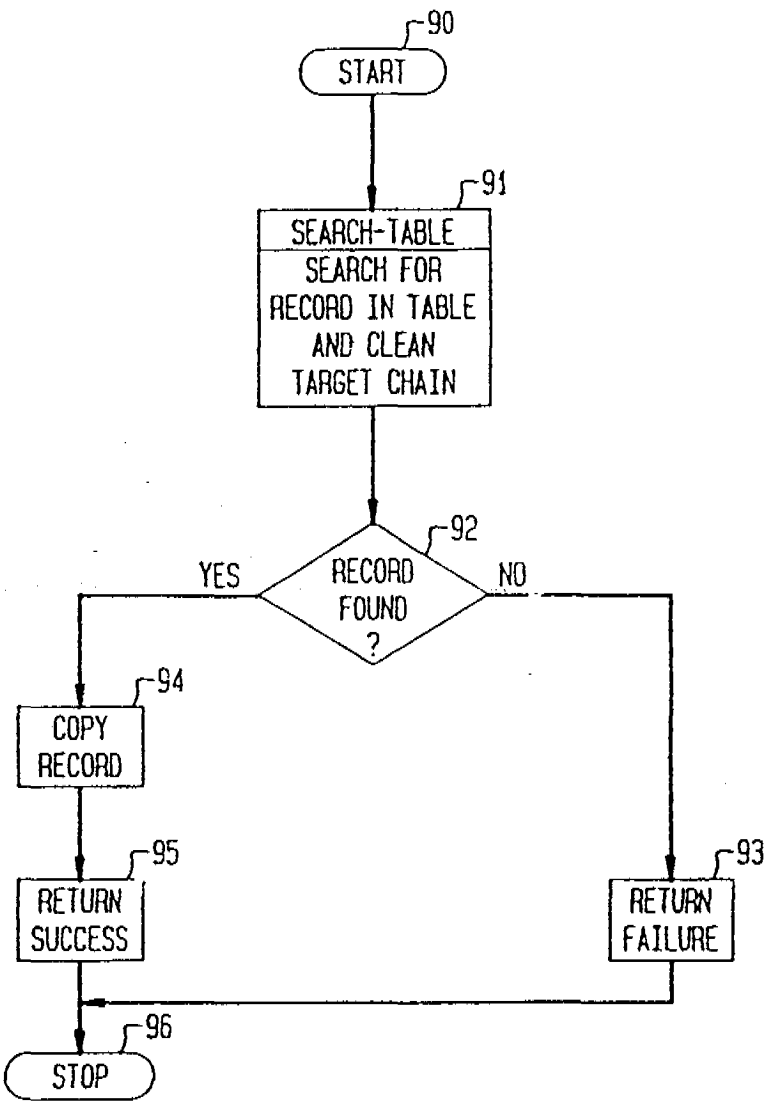
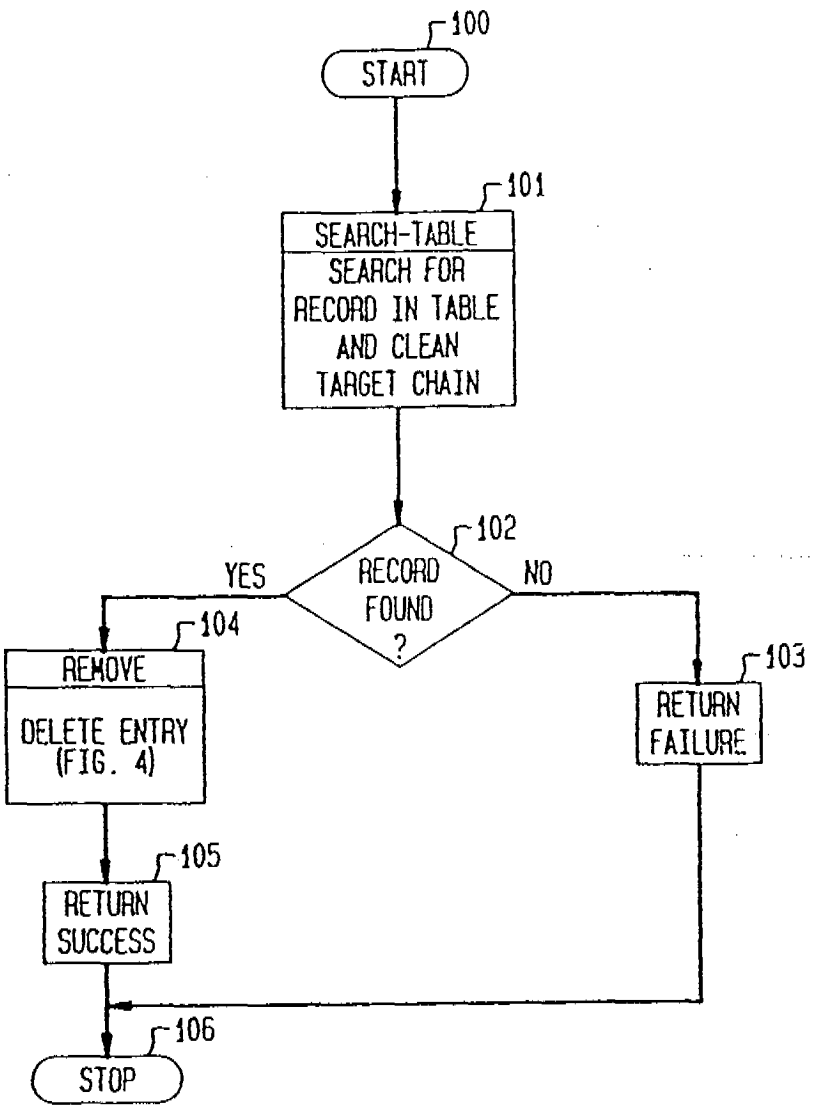

90
START
91
SEARCH-TABLE
SEARCH FOR RECORD IN TABLE AND CLEAN TARGET CHAIN
92
RECORD FOUND ?
YES
NO
94
COPY RECORD
95
RETURN SUCCESS
93
RETURN FAILURE
96
STOP

FIG. 7



METHODS AND APPARATUS FOR INFORMATION STORAGE AND RETRIEVAL UTILIZING HASHING TECHNIQUES

This application is a continuation of application Ser. No. 07/151,639, filed Feb. 2, 1988 now abandoned.

TECHNICAL FIELD

This invention relates to information storage and retrieval systems and, more particularly, to the use of hashing techniques in such systems.

BACKGROUND OF THE INVENTION

Information or data stored in a computer-controlled storage mechanism can be retrieved by searching for a particular key in the stored records. The stored record with a key matching the search key is then retrieved. Such searching techniques require repeated accesses or probes into the storage mechanism to perform key comparisons. In large storage and retrieval systems, such searching, even if augmented by efficient search algorithms such as a binary search, often requires an excessive amount of time.

Another well-known and much faster method for storing and retrieving information from computer store involves the use of so-called "hashing" techniques. These techniques are also sometimes called scatter-storage or key-transformation techniques. In a system using hashing, the key is operated upon (by a hashing function) to produce a storage address in the storage space (called the hash table). This storage address is then used to access the desired storage location directly with fewer storage accesses or probes than sequential or binary searches. Hashing techniques are described in the classic text by D. Knuth entitled *The Art of Computer Programming*, Volume 3, *Sorting and Searching*, pp. 506–549, Addison-Wesley, Reading, Mass., 1973.

Hashing functions are designed to translate the universe of keys into addresses uniformly distributed throughout the hash table. Typical hashing operations include truncation, folding, transposition and modulo arithmetic. A disadvantage of hashing techniques is that more than one key can translate into the same storage address, causing "collisions" in storage or retrieval operations. Some form of collision-resolution strategy (sometimes called "rehashing") must therefore be provided. For example, the simple strategy of searching forward from the initial storage address to the first empty storage location will resolve the collision. This latter technique is called linear probing. If the hash table is considered to be circular so that addresses beyond the end of the table map back to the beginning of the table, then the linear probing is done with "open addressing," i.e., with the entire hash table as overflow space in the event that a collision occurs.

Some forms of data records have a limited lifetime after which they become obsolete. Scheduling activities, for example, involves records which become obsolete after the scheduled activity has occurred. Such record storage locations cannot be simply emptied since this location may be a link in a chain of locations previously created during a collision-resolution procedure. The classic solution to this problem is to mark the record as "deleted" rather than as "empty," and to leave the record in place. In time, however, the storage space can become contaminated by an excessive number of deleted or obsolete storage locations that must be searched to locate desired records. With the passage of time, such storage contamination can reduce the performance of retrieval operations below acceptable levels. Problems of this type are discussed in considerable detail in *Data Structures and Program Design*, by R. L. Kruse, Prentice-Hall, Englewood Cliffs, N.J., 1984, pp. 112–126, and *Data Structures with Abstract Data Types and PASCAL*, by D. F. Stubbs and N. W. Webre, Brooks/Cole Publishing, Monterey, Calif., 1985, pp. 310–336.

In the prior art, such storage space contamination was avoided by deletion procedures that eliminated deleted records by replacing the deleted record with another record in the collision-resolution chain of records and thus close the chain without leaving any deleted records. One such procedure is shown in the aforementioned text by Knuth at page 527. Unfortunately, such non-contaminating procedures, due to the necessity for successive probes into the storage space, take so much time that they can be used only when the data base is off line and hence not available for accessing.

The problem, then, is to provide the speed of access of hashing techniques for large and heavily used information storage systems having expiring data and, at the same time, prevent the large-scale contamination which normally results from expired records in such large and heavily used systems.

SUMMARY OF THE INVENTION

In accordance with the illustrative embodiment of the invention, these and other problems are overcome by using a garbage collection procedure "on the fly" while other types of access to the storage space are taking place. In particular, during normal data insertion or retrieval probes into the data store, the expired, obsolete records are identified and removed in the neighborhood of the probe. Specifically, expired or obsolete records in the collision-resolution chain including the record to be accessed are removed as part of the normal retrieval procedure.

This incremental garbage collection technique has the decided advantage of automatically eliminating contamination caused by obsolete or expired records without requiring that the data base be taken off-line for such garbage collection. This is particularly important for data bases requiring rapid access and continuous availability to the user population.

BRIEF DESCRIPTION OF THE DRAWING

A complete understanding of the present invention may be gained by considering the following detailed description in conjunction with the accompanying drawing, in which:

FIG. 1 shows a general block diagram of a computer system hardware arrangement in which the information storage and retrieval system of the present invention might be implemented;

FIG. 2 shows a general block diagram of a computer system software arrangement in which the information storage and retrieval system of the present invention might find use;

FIG. 3 shows a general flow chart for table searching operation which might be used in a hashed storage system in accordance with the present invention;

FIG. 4 shows a general flow chart for a garbage collecting remove procedure which forms part of the table searching operation of FIG. 3;

FIG. 5 shows a general flow chart for record insertion operations which might be used in a hashed storage system in accordance with the present invention;

FIG. 6 shows a general flow chart for a record retrieval operation for use in a hashed storage system in accordance with the present invention; and

FIG. 7 shows a general flow chart for a record deletion operation which might be used in the hashed storage system in accordance with the present invention.

To facilitate reader understanding, identical reference numerals are used to designate elements common to the figures.

DETAILED DESCRIPTION

Referring more particularly to FIG. 1 of the drawings, there is shown a general block diagram of a computer hardware system comprising a Central Processing Unit (CPU) 10 and a Random Access Memory (RAM) unit 11. Computer programs stored in the RAM 11 are accessed by CPU 10 and executed, one instruction at a time, by CPU 10. Data, stored in other portions of RAM 11, are operated upon by the program instructions accessed by CPU 10 from RAM 11, all in accordance with well-known data processing techniques.

Central Processing Unit (CPU) 10 also controls and accesses a disk controller unit 12 which, in turn, accesses digital data stored on one or more disk storage units such as disk storage unit 13. In normal operation, programs and data are stored on disk storage unit 13 until required by CPU 10. At this time, such programs and data are retrieved from disk storage unit 13 in blocks and stored in RAM 11 for rapid access.

Central Processing Unit (CPU) 10 also controls an Input-Output (IO) controller 14 which, in turn, provides access to a plurality of input devices such as CRT (cathode ray tube) terminal 15, as well as a plurality of output devices such as printer 16. Terminal 15 provides a mechanism for a computer operator to introduce instructions and commands into the computer system of FIG. 1, and may be supplemented with other input devices such as card and tape readers, remotely located terminals, optical readers and other types of input devices. Similarly, printer 16 provides a mechanism for displaying the results of the operation of the computer system of FIG. 1 for the computer user. Printer 16 may similarly be supplemented by line printers, cathode ray tube displays, phototypesetters, graphical plotters and other types of output devices.

The constituents of the computer system of FIG. 1 and their cooperative operation are well-known in the art and are typical of all computer systems, from small personal computers to large main frame systems. The architecture and operation of such systems are well-known and, since they form no part of the present invention, will not be further described here.

In FIG. 2 there is shown a graphical representation of a typical software architecture for a computer system such as that shown in FIG. 1. The software of FIG. 2 comprises an access mechanism 20 which, for simple personal computers, may comprise no more than turning the system on. In larger systems, providing service to a larger number of users, login and password procedures would typically be implemented in access mechanism 20. Once access mechanism 20 has completed the login procedure, the user is placed in the operating system environment 21. Operating system 21 coordinates the activities of all of the hardware components of the computer system (shown in FIG. 1) and provides a number of utility programs 22 of general use to the computer user. Utilities 22 might, for example, comprise assemblers and compilers, mathematical routines, basic file handling routines and system maintenance facilities.

Many computer software systems also include a data base manager program 23 which controls access to the data records in a data base 24. Data base 24 may, for example, reside on a disk storage unit or units such as disk storage unit 13 of FIG. 1. User application programs such as application program 25 then use the data base manager program 23 to access data base records in data base 24 for adding, deleting and modifying data records. It is the efficient realization of a data base manager such as data base manager program 23 in FIG. 2 to which the present invention is directed.

Before proceeding to a description of one embodiment of the present invention, it is first useful to discuss hashing techniques in general. Hashing techniques have been used classically for very fast access to static, short term data such as a compiler symbol table. Typically, in such storage tables, deletions are infrequent and the need for the storage table disappears quickly.

In some common types of data storage systems, data records become obsolete merely by the passage of time or by the occurrence of some event. If such expired, lapsed or obsolete records are not removed from the storage table, they will, in time, seriously degrade or contaminate the performance of the retrieval system. Contamination arises because of the ever-increasing need to search longer and longer chains of record locations, many of which are expired, to reach a desired location.

More particularly, a hash table can be described as a logically contiguous, circular list of consecutively numbered, fixed-sized storage units, called cells, each capable of storing a single item called a record. Each record contains a distinguishing field, called the key, which is used as the basis for storing and retrieving the associated record. The keys throughout the hash table data base are distinct and unique for each record. Hashing functions which associate keys with storage addresses are usually not one-to-one in that they map many distinct keys into the same location.

To store a new record, a cell number is generated by invoking the hashing function on the key for the new record. If this cell location is not occupied, the new record is stored there. If this cell location is occupied, a collision has occurred and the new record must be stored elsewhere, in an overflow area, using an appropriate collision-resolution technique. A common collision-resolution strategy, which will be described here, is known as linear probing under open addressing. Open addressing means that the overflow area is the entire hash table itself. Linear probing indicates sequential scanning of cells beginning with the next cell, recalling that the storage table is viewed circularly. The collision is resolved by storing the record in the first unoccupied cell found.

To retrieve a record, the key is hashed to generate a cell location. If the record is not there (the keys do not match), searching continues following the same forward path as record storage. An empty cell terminates the retrieval procedure, which has then failed to find the record to be retrieved.

In FIG. 3 there is shown a flowchart of a *search table* procedure for searching the hash table preparatory to inserting, retrieving or deleting a record. The hash table may, for example, comprise the data base 24 of FIG. 2

and the *search table* procedure of FIG. 3 comprise a portion of the data base manager 23 of FIG. 2. Starting in box 30 of the *search table* procedure of FIG. 3, the search key of the record being searched for is hashed in box 31 to provide the address of a cell. In box 32, the empty cell just past the end of the search chain of non-empty cells is located, i.e., the first succeeding unoccupied cell is found. In box 33, the procedure moves one cell backward from the current cell position (now at the end of the chain). Decision box 34 examines the cell to determine whether the cell is empty or not. If the cell tested in decision box 34 is empty, decision box 35 is entered to determine if a key match was previously found in decision box 41 (as will be described below). If so, the search is successful and returns success in box 36 and terminates in terminal box 39. If not, box 37 is entered where the location of the empty cell is saved for possible record insertion. In box 38 failure is returned since an empty cell was found before a cell with a matching key. The procedure again terminates in box 39.

If the cell tested in decision box 34 is not empty, decision box 40 is entered to determine if the record in that cell has expired. This is determined by comparing some portion of the contents of the record to some external condition. A timestamp in the record, for example, could be compared with the time-of-day. Alternatively, the occurrence of an event can be compared with a field identifying that event in the record. In any event, if the record has not expired, decision box 41 is entered to determine if the key in this record matches the search key. If it does, the cell location is saved in box 42 and the procedure returns to box 33. If the record key does not match the search key, the procedure returns directly to box 33.

If decision box 40 determines that the record has expired, box 43 is entered to perform a non-contaminating deletion of the expired record, as will be described in connection with FIG. 4. In general, the procedure of box 43 (FIG. 4) operates to move a record further toward the end of the chain into the position of the record which has expired, thereby removing the expired record and, at the same time, closing the search chain.

It can be seen that the *search table* procedure of FIG. 3 operates to examine the entire chain of records of which the searched-for record is a part, and to delete expired records by chain-filling rather than by marking such records as deleted. In this way, contamination of the storage space by expired records is removed in the vicinity of each new table search. If contamination becomes too large even with such automatic garbage collection, then the insertion of new records can be inhibited until the *search table* procedure has had a chance to remove a sufficient number of expired records to render the operation of the system sufficiently efficient.

The *search table* procedure illustrated generally in FIG. 3 is implemented in the Appendix as PASCAL-like pseudocode. Source code suitable for compilation and execution on any standard hardware and software computing system can readily be devised from this pseudocode and the flowcharts of the figures by any person of ordinary skill in the art.

In FIG. 4 there is shown a flowchart of a *remove* procedure which removes records from the database, either records to be deleted or expired records. In general, this is accomplished by traversing the chain of the record to be removed in a forward direction searching for a record whose key hashes at or behind the cell to be removed. When such a record is found, it is copied to the cell of the record to be removed. The copied record is then taken as the record to be removed and the process is continued until the end of the search chain is reached. In box 54, the final copied record is marked empty prior to teminating the procedure. The *remove* procedure of FIG. 4 might comprise a portion of the data base manager program 23 of FIG. 2.

Starting at starting box 50 of FIG. 4, the procedure is entered with the location of a cell to be removed which is called the base cell. Initially, box 51 is entered where the load count in the table is adjusted to reflect the removal of one record. The load, of course, is the number of occupied cells. As previously noted, the value of this load can be used to disable the insertion of new records until the load has reached a low enough value to permit efficient searching. In box 52, the procedure of FIG. 4 advances to the next cell in the chain beyond the base cell. In decision box 53 this cell is tested to see if it is empty. If it is empty, the end of the chain has been reached and box 54 is entered to mark the base cell as empty. Decision box 55 is then entered to determine if a record was found (by the *search table* procedure) which matched the search key and, if so, the procedure is terminated in terminal box 56. If a matching record was not found, decision box 57 is entered to determine if the base cell is ahead of the hash location of the search key. If not, the procedure is terminated in box 56. If the base cell does hash ahead of the search record, then the base cell can be used for storing a new record. In box 58, the location of this empty cell is therefore saved as a possible insertion site.

Returning to box 53, if the next cell is not empty, box 59 is entered to determine if the record in this cell hashes ahead of the base cell. If so, box 52 is re-entered to advance to the next cell in the chain. If this next cell hashes at or behind the base cell, however, box 60 is entered to copy the contents of this next cell to the base cell, thereby obliterating (removing) the base cell contents. Box 61 is then entered to test if the *search table* procedure found a matching record. If not, box 52 is re-entered to advance to the next cell. If a matching record was found, decision box 62 is entered to test if the matching record is the base cell record. If not, box 52 is re-entered to advance to the next cell. If the matching record is the base cell, however, box 63 is entered to store the location of the former base cell as the position of the matching record and then box 52 is re-entered to advance to the next cell in the search chain.

It can be seen that the procedure of FIG. 4 operates to examine the entire search chain and to move records from later positions in the chain to vacated positions in the chain such that the chain is entirely closed at the end of the procedure. That is, no empty cells are left to erroneously break up a search chain. As noted in connection with FIG. 3, expired records are subjected to the *remove* procedure of FIG. 4. As will be noted in connection with FIG. 7, records to be deleted from the data base are also subjected to the *remove* procedure of FIG. 4.

The *remove* procedure illustrated generally in FIG. 4 is implemented in the Appendix as PASCAL-like pseudocode. Source code suitable for compilation and execution on any standard hardware and software computing system can readily be devised from this pseudo-

code and the flowchart of FIG. 4 by any person of ordinary skill in the art.

In FIG. 5 there is shown a detailed flowchart of an *insert* procedure suitable for use in the information storage and retrieval system of the present invention. The *insert* procedure of FIG. 5 begins as starting box 70 from which box 71 is entered. In box 71, the *search table* procedure of FIG. 3 is invoked with the search key of the record to be inserted. As noted in connection with FIG. 3, the *search table* procedure locates the target cell location and, if part of a search chain, removes all expired cells from that search chain. Decision box 72 is then entered where it is determined whether or not the *search table* procedure found a record with a matching key. If so, box 73 is entered where the record to be inserted in put into the storage table in the position of the old record with a matching key. In box 74, the *insert* procedure reports that the old record has been replaced by the new record and the procedure is terminated in terminal box 75.

Returning to decision box 72, if a matching record is not found, decision box 76 is entered to determine if the table load is below a preselected threshold (typically about 75% of the table capacity). If the load is not below the threshold, the storage table is too full to be access efficiently, and box 77 is entered to report that the the table is full and the record cannot be inserted. The procedure then terminates in terminal box 75. If the load is below the threshold, box 78 is entered where the record to be inserted is placed in the empty cell position found by the *search table* procedure. In box 79, the load is adjusted to reflect the addition of one record to the storage table, the procedure reports that the record was inserted in box 80 and the procedure terminated in box 75.

The *insert* procedure illustrated generally in FIG. 5 is implemented in the Appendix as PASCAL-like pseudocode. Source code suitable for compilation and execution on any standard hardware and software computing system can readily be devised from this pseudocode and the flowcharts of the FIG. 5 by any person of ordinary skill in the art.

In FIG. 6 there is show a detailed flowchart of a *retrieve* procedure which is used to retrieve a record from the data base 24 of FIG. 2. Starting in box 90, the *search table* procedure is invoked in box 91, using the key of the record to be retrieved as the search key. In box 92 it is determined if a record with a matching key was found by the *search table* procedure. If not, box 93 is entered to report failure of the *retrieve* procedure and the procedure is terminated in box 96. If a matching record was found, box 94 is entered to copy the matching record into a buffer store for processing by the calling program, box 95 is entered to return an indication of successful retrieval and the procedure terminated in box 96.

The pseudo-code for the *retrieve* procedure of FIG. 6 is included in the Appendix. Executable code for all common hardware and system software arrangements can readily be devised by those skilled in the art from the flowchart and the pseudo-code.

In FIG. 7 there is shown a detailed flowchart of a *delete* procedure useful for actively removing records from the data base 24 of FIG. 2. Starting at box 100, the procedure of FIG. 7 first invokes the *search table* procedure of FIG. 3 ih box 101, using the key of the record to be deleted as the search key. In box 102, it is determined if the *search table* procedure was able to locate a record with a matching key. If not, box 103 is entered to report failure of the deletion procedure and the procedure is terminated in box 106. If a matching record was found, as determined by box 102, the *remove* procedure of FIG. 4 is invoked in box 104. As noted in connection with FIG. 4, this procedure removes the record to be deleted and, at the same time, closes the search chain. Box 105 is then entered to report successful deletion to the calling program and the procedure is terminated in box 106.

The *delete* procedure illustrated generally in FIG. 7 is implemented in the Appendix as PASCAL-like pseudocode. Source code suitable for compilation and execution on any standard hardware and software computing system can readily be devised from this pseudocode and the flowchart of FIG. 7 by any person of ordinary skill in the art.

The attached Appendix contains pseudocode listings for all of the programmed functions necessary to implement a data base manager 23 (FIG. 2) operating in accordance with the present invention. These listings follow the flowcharts of FIGS. 3–7 and further explain and elucidate the flowcharts. Any person of ordinary skill in the art will have no difficulty implementing these functions in any desired program language to run on any desired computer hardware configuration.

It should also be clear to those skilled in the art that further embodiments of the present invention may be made by those skilled in the art without departing from the teachings of the present invention.

APPENDIX

```
Functions Provided

The following functions are made available to the
application program:

insert (record: record type)

Returns replaced if a record associated with record.key
was found in the table and subsequently replaced.

Returns inserted if a record associated with record.key
was not found in the table and the passed record was
subsequently inserted.

Returns full if a record associated with record.key was
not found in the table and passed record could not be
inserted because load factor has reached max load factor.

retrieve (record: record type)

Returns success if record associated with record.key was
found in the table and assigned to record.

Returns failure if search was unsuccessful.

delete (record key: record key type)

Returns success if record associated with record key was
found in the table and subsequently deleted

Returns failure if none found.

Definitions

The following formal definitions are required for
specifying the insertion, retrieval, and deletion
algorithms:

const table size          /* size of hash table */
```

```
const max load factor     /* 0 ≤ max load factor < 1 */

var table: array[0 .. table size-1] of record type:
                /* hash table */

var load: 0 .. table size-1;
                /* number of occupied entries of
                   hash table array (initially 0) */
```

Algorithms

Algorithms for the functions described above are given below:

```
function insert (record: record type):
        (replaced, inserted, full );

  var position: 0 .. table size-1;
                /* position in table to update or
                   insert (returned by search table) */

  begin

    if search table (record.key, position)

       then begin

         table[position] := record;
         return (replaced)

       end

       else if load/table size < max load factor

         then begin

           load := load+1;
           table[position] := record;
           return (inserted)

         end

         else return (full)

  end               /* insert */

  function retrieve (var record: record type):
          (success, failure);

    var position: 0 .. table size-1;
              /* position in table where record
                 resides (returned by search table) */

    begin

      if search table (record.key, position)

         then begin

           record := table[position];
           return (success)

         end

         else return (failure)

    end  /* retrieve */

  function delete (record key: record key type):
         (success, failure);

    var position: 0 .. table size-1;
              /* position in table where record
                 resides (returned by search table) */
    dummy variable: 0 .. table size-1;
              /* last two arguments to remove are
                 not relevant here                  */

    begin

      if search table (record key, position)

         then begin

           remove (position, true, dummy variable,
                   dummy variable);

           return (success)

         end

         else return (failure)

    end    /* delete */

function search table (record key: record key type;
        var position: 0 .. table size-1): boolean;

 /* search table for record key and delete expired
    expired records in target chain; position is set to
    index of found record or appropriate empty cell */

  var i: 0 .. table size-1;
       /* used for scanning chain,
          both forwards & backwards */

      pos empty: 0 .. table size-1;
       /* index of leftmost empty cell
          to right of position */

      is rec found: boolean;
       /* indicates whether search is successful */


  begin

    position := hash (record key);
    is rec found := false;

    if table[position] is not empty then


      begin
        i := position; /* loop initialization */
        repeat         /* scan forward to end of chain
                          containing table[position]  */

          i := (i+1) mod table size

        until (table[i] is empty);

        pos empty := i;
      i := (i-1+table size) mod table size;

      while (table[i] is not empty) do
            /*scan chain in reverse,
              deleting expired entries */

        begin

          if table[i] is expired
                   then remove (i, is rec found,
                                position, pos empty)

                   else if table[i].key = record key
```

```
                then begin

                    is rec found := true;
                    position := i

                end;

            i := (i-1+table size) mod table size

        end;   /* while */

    if not is rec found then position := pos empty

  end;    /* then */

 return (is rec found)

end                    /* search table  */

procedure remove (cell to del:
 0 .. table size-1; is rec found: boolean;
 var pos of search rec, pos empty: 0 .. table size-1;

          /* Delete table[cell to del] */

var i, j: 0 .. table size-1;

begin

  load := load-1;

  do forever

    i := cell to del;
            /* save position of emptied slot */

    repeat          /* scan forward looking for a
                       record to fill hole in chain */

      cell to del := (cell to del+1) mod table size;

      if table[cell to del] is empty

        then begin

          table[i] := empty;

          if not is rec found then

            if (pos of search rec ≤ i < pos empty)
            or (i< pos empty < pos of search rec)
            or (pos empty< pos of search rec ≤
            i) then pos empty := i;

            return
          end;

      j := hash (table[cell to del].key)

    until (j ≤ i < cell to del)
    or (i < cell to del < j)
    or (cell to del< j ≤ i);

    table[i] := table[cell to del];
    /* use table[cell to del] to plug hole in chain */

    if (is rec found) and
    (pos of search rec = cell to del)
    then pos of search rec := i

  end

end                    /* remove */
```

What is claimed is:

1. An information storage and retrieval system using hashing techniques to provide rapid access to the records of said system and utilizing a linear probing technique to store records with the same hash address, at least some of said records automatically expiring, said system comprising

a record search means utilizing a search key to access a chain of records having the same hash address,

said record search means including means for identifying and removing all expired ones of said records from said chain of records each time said chain is accessed, and

means, utilizing said record search means, for inserting retrieving and deleting records from said system and, at the same time, removing all expired ones of said records in the accessed chains of records.

2. The information storage and retrieval system according to claim 1 further comprising

means for recursively moving a record from a later position in said chain of records into the position of one of said expired records.

3. The information storage and retrieval system according to claim 1 further including

means for counting the number of records in said system,

means, responsive to said counting means, for inhibiting the insertion of new records into said system when the number of records in said system exceeds a preselected value.

4. The information storage and retrieval system according to claim 3 further including

means, also responsive to said counting means, for re-enabling the insertion of new records into said system when the number of records in said system falls below said preselected value.

5. A method for storing and retrieving information records using hashing techniques to provide rapid access to said records and utilizing a linear probing technique to store records with the same hash address, at least some of said records automatically expiring, said method comprising the steps of

accessing a chain or records having the same hash address,

identifying the automatically expired ones of said records,

removing all automatically expired records from said chain of records each time said chain is accessed, and

inserting, retrieving or deleting one of said records from said system following said step of removing.

6. The method according to claim 5 further comprising the step of

moving a record from a later position in said chain of records into the position of one of said expired records.

7. The method according to claim 5 further comprising the steps of

counting the number of records in said system, and

inhibiting the insertion of new records into said system when the number of records in said system rises above a preselected value.

8. The method according to claim 7 further comprising the step of

re-enabling the insertion of new records into said system when the number of records in said system falls below said preselected value.

* * * * *

US005121495A

United States Patent [19]

Nemes

[11] Patent Number: 5,121,495

[45] Date of Patent: Jun. 9, 1992

[54] METHODS AND APPARATUS FOR INFORMATION STORAGE AND RETRIEVAL UTILIZING HASHING TECHNIQUES

[75] Inventor: Richard M. Nemes, Brooklyn, N.Y.

[73] Assignee: Bell Communications Research, Inc., Livingston, N.J.

[21] Appl. No.: 430,901

[22] Filed: Oct. 31, 1989

Related U.S. Application Data

[63] Continuation of Ser. No. 151,639, Feb. 2, 1988, abandoned.

[51] Int. Cl.[5] G06F 15/411; G06F 12/00

[52] U.S. Cl. 395/600; 364/222.81; 364/222.82; 364/281.1; 364/282.1; 364/252.3; 364/DIG. 1

[58] Field of Search ... 364/200 MS File, 900 MS File

[56] References Cited

U.S. PATENT DOCUMENTS

4,121,236	10/1978	Venton et al.	364/900 X
4,215,402	7/1980	Mitchell et al.	364/200
4,447,875	5/1984	Bolton et al.	364/200
4,502,118	2/1985	Hagenmaier, Jr. et al.	364/200
4,716,524	12/1987	Oxley et al.	364/200
4,775,932	10/1988	Oxley et al.	364/200

OTHER PUBLICATIONS

"The Art of Computer Programming", Sorting and Searching, D. E. Knuth, Addison-Wesley Series in Computer Science and Information Processing, pp. 506–549, 1973.

"Data Structures with Abstract Data Types and Pascal", D. F. Stubbs and N. W. Webre, Brooks/Cole Publishing Company, 1985, Section 7.4, Hashed Implementations, pp. 310–336.

"Data Structures and Program Design", R. L. Kruse, Prentice-Hall, Inc. 1984, Section 3.7, Hashing, pp. 112–126.

Primary Examiner—Gareth D. Shaw
Assistant Examiner—Paul Kulik
Attorney, Agent, or Firm—James W. Falk

[57] ABSTRACT

A method and apparatus for performing storage and retrieval in an information storage system is disclosed which uses the hashing technique. In order to prevent contamination of the storage medium by automatically expiring records, a garbage collection technique is used which removes all expired records in the neighborhood of a probe into the data storge system. More particularly, each probe for insertion, retrieval or deletion of a record is an occasion to search the entire chain of records found for expired records and then removing them and closing the chain. This garbage collection automatically removes expired record contamination in the vicinity of the probe, thereby automatically decontaminating the storage space. Because no long term contamination can build up in the present system, it is useful for large data bases which are heavily used and which require the fast access provided by hashing.

8 Claims, 6 Drawing Sheets

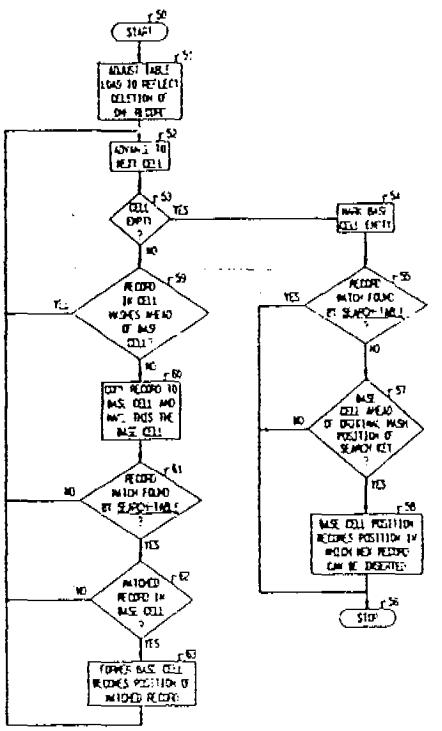


FIG. 1

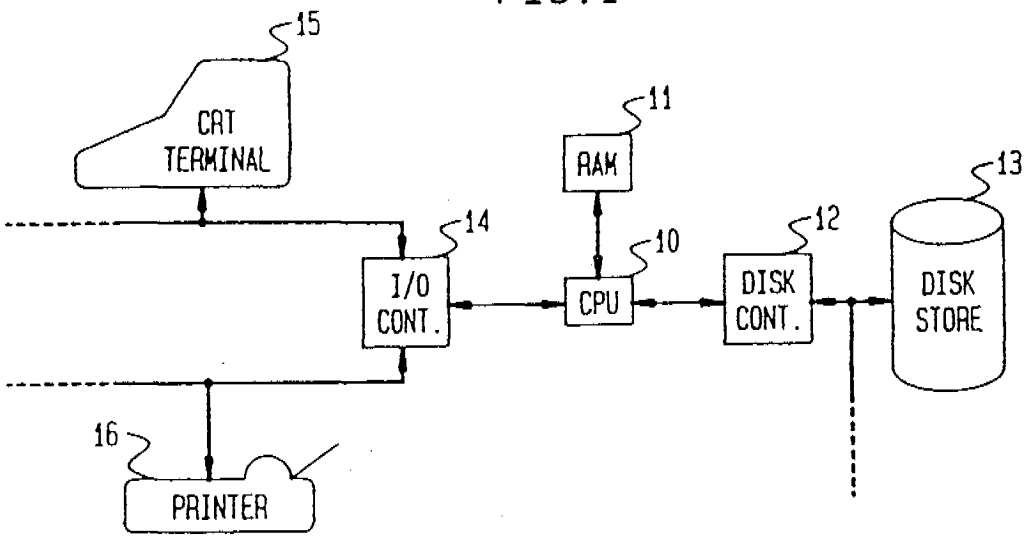


FIG. 2

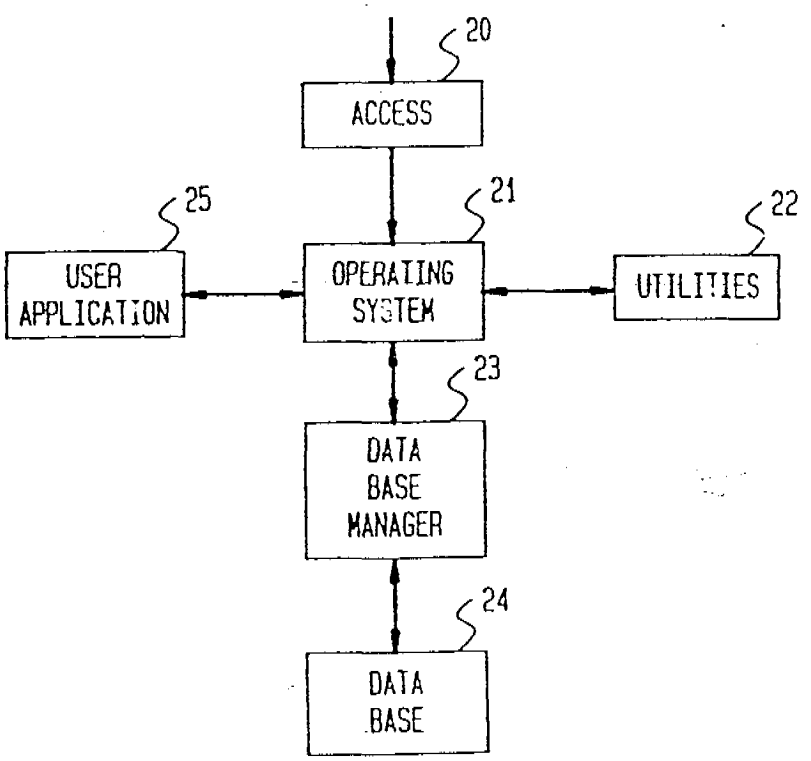


FIG. 3

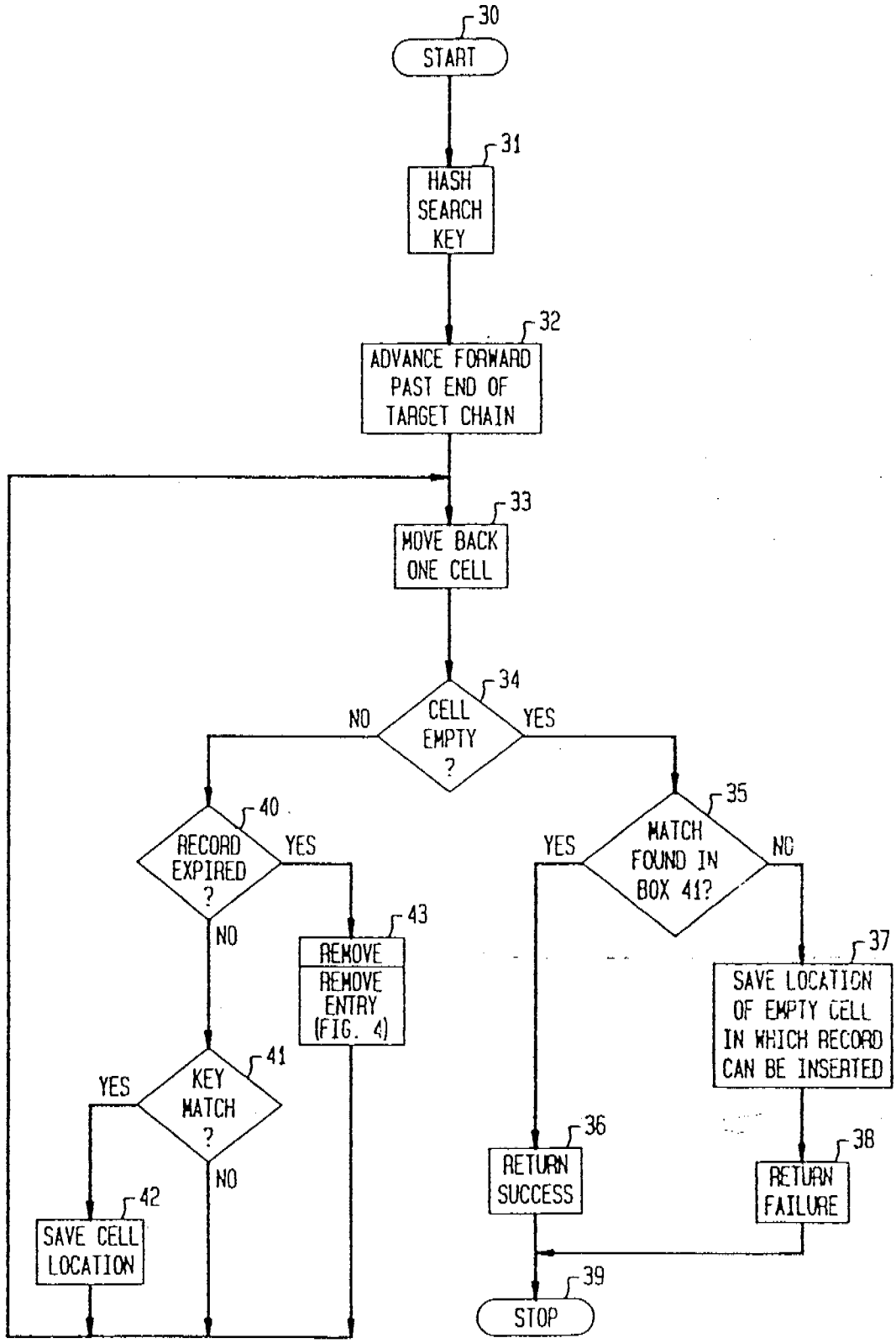


FIG. 4

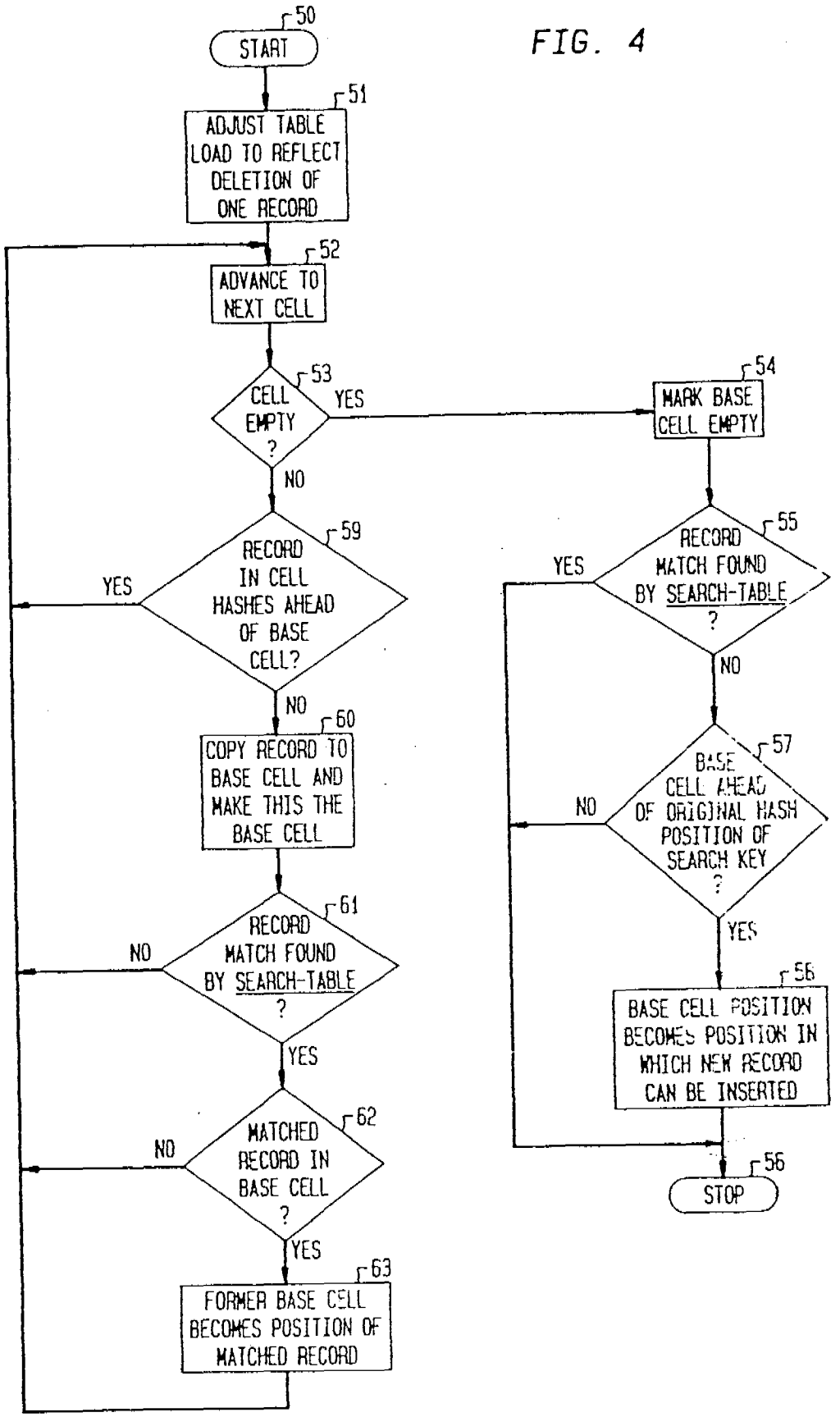


FIG. 5

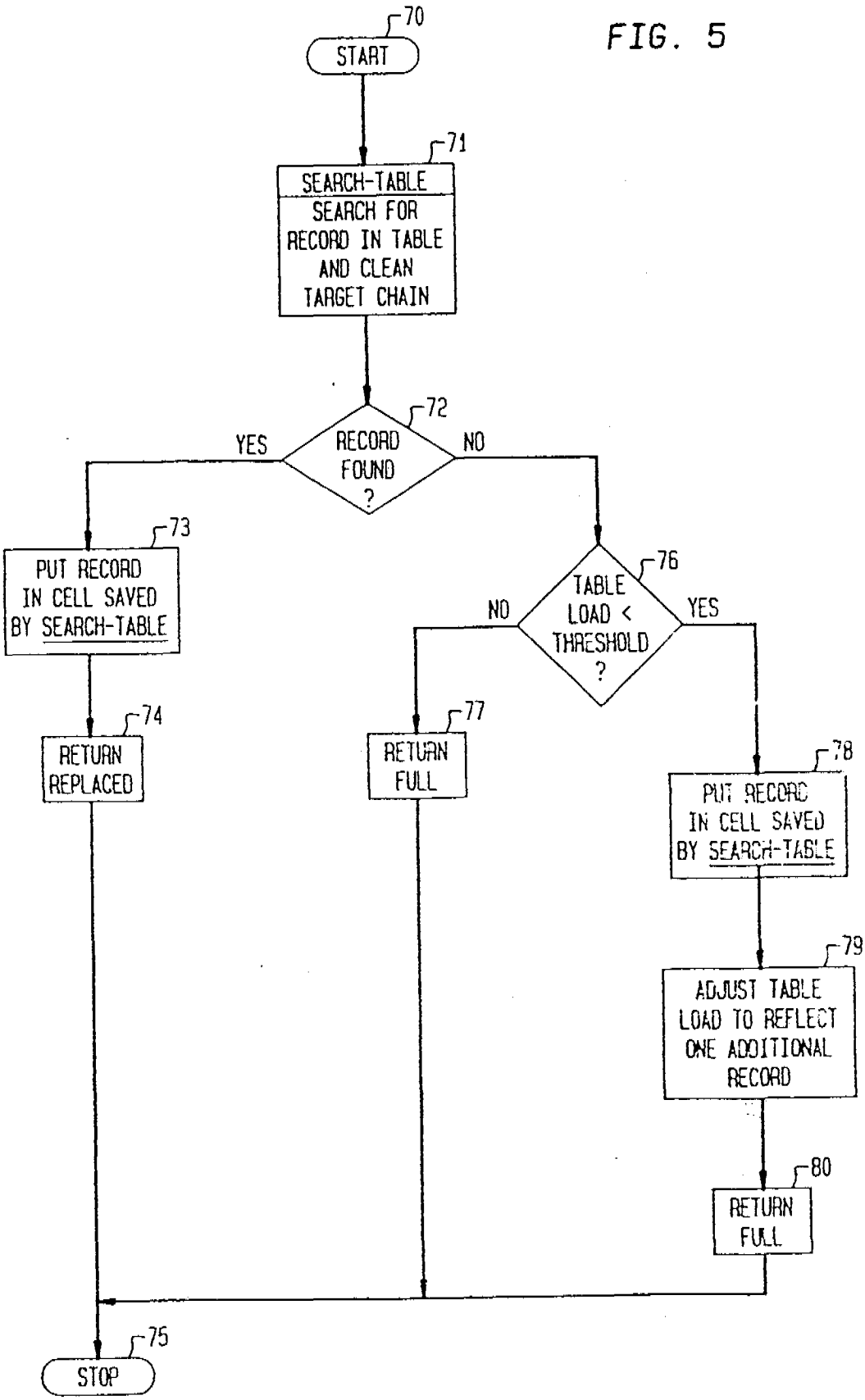


FIG. 6

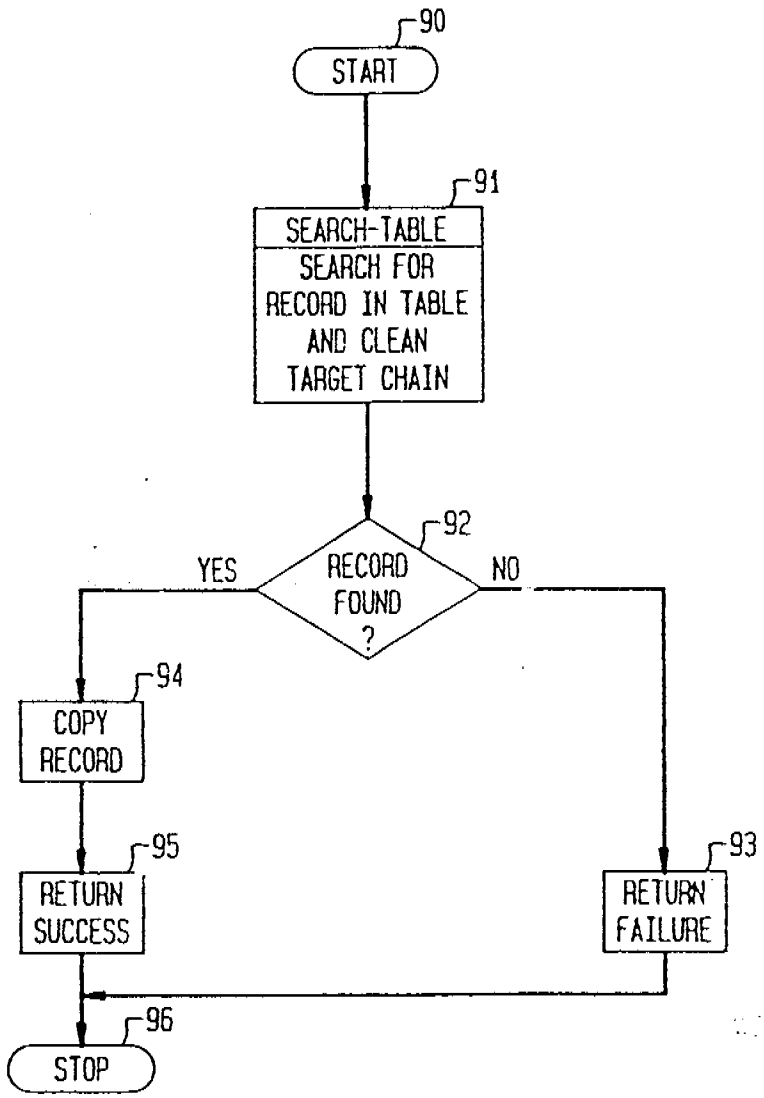
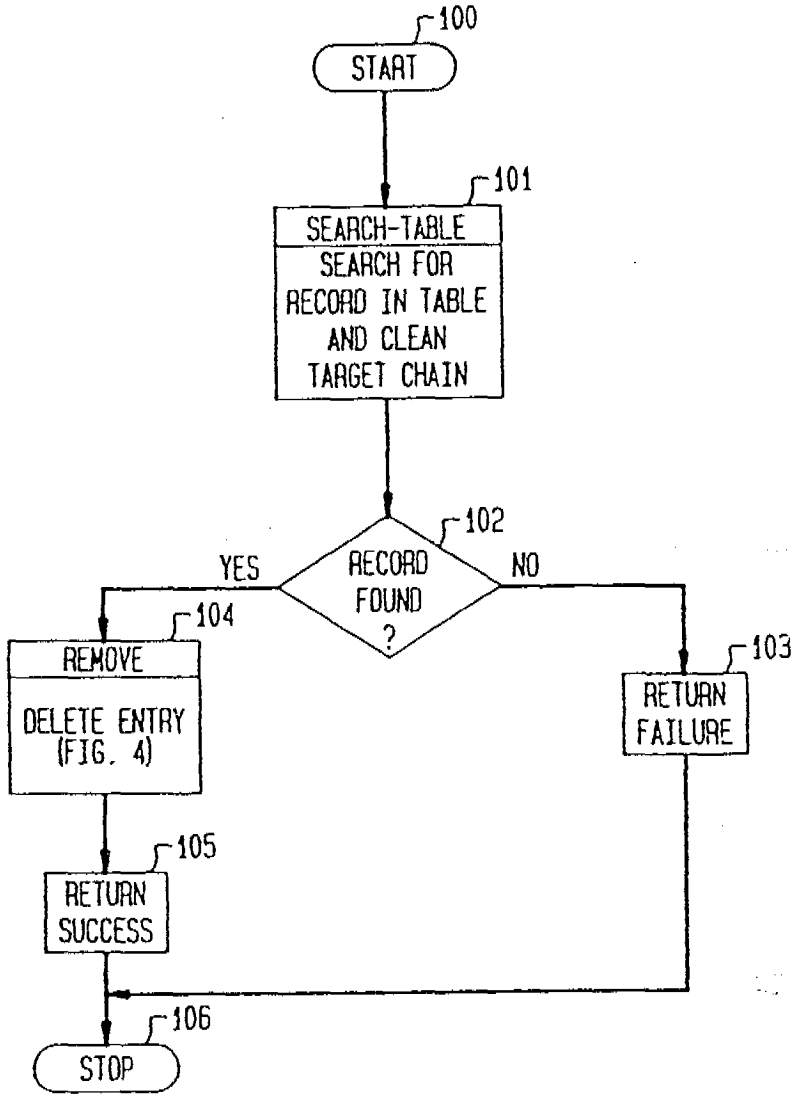


FIG. 7



METHODS AND APPARATUS FOR INFORMATION STORAGE AND RETRIEVAL UTILIZING HASHING TECHNIQUES

This application is a continuation of application Ser. No. 07/151,639, filed Feb. 2, 1988 now abandoned.

TECHNICAL FIELD

This invention relates to information storage and retrieval systems and, more particularly, to the use of hashing techniques in such systems.

BACKGROUND OF THE INVENTION

Information or data stored in a computer-controlled storage mechanism can be retrieved by searching for a particular key in the stored records. The stored record with a key matching the search key is then retrieved. Such searching techniques require repeated accesses or probes into the storage mechanism to perform key comparisons. In large storage and retrieval systems, such searching, even if augmented by efficient search algorithms such as a binary search, often requires an excessive amount of time.

Another well-known and much faster method for storing and retrieving information from computer store involves the use of so-called "hashing" techniques. These techniques are also sometimes called scatter-storage or key-transformation techniques. In a system using hashing, the key is operated upon (by a hashing function) to produce a storage address in the storage space (called the hash table). This storage address is then used to access the desired storage location directly with fewer storage accesses or probes than sequential or binary searches. Hashing techniques are described in the classic text by D. Knuth entitled *The Art of Computer Programming*, Volume 3, *Sorting and Searching*, pp. 506–549, Addison-Wesley, Reading, Mass., 1973.

Hashing functions are designed to translate the universe of keys into addresses uniformly distributed throughout the hash table. Typical hashing operations include truncation, folding, transposition and modulo arithmetic. A disadvantage of hashing techniques is that more than one key can translate into the same storage address, causing "collisions" in storage or retrieval operations. Some form of collision-resolution strategy (sometimes called "rehashing") must therefore be provided. For example, the simple strategy of searching forward from the initial storage address to the first empty storage location will resolve the collision. This latter technique is called linear probing. If the hash table is considered to be circular so that addresses beyond the end of the table map back to the beginning of the table, then the linear probing is done with "open addressing," i.e., with the entire hash table as overflow space in the event that a collision occurs.

Some forms of data records have a limited lifetime after which they become obsolete. Scheduling activities, for example, involves records which become obsolete after the scheduled activity has occurred. Such record storage locations cannot be simply emptied since this location may be a link in a chain of locations previously created during a collision-resolution procedure. The classic solution to this problem is to mark the record as "deleted" rather than as "empty," and to leave the record in place. In time, however, the storage space can become contaminated by an excessive number of deleted or obsolete storage locations that must be searched to locate desired records. With the passage of time, such storage contamination can reduce the performance of retrieval operations below acceptable levels. Problems of this type are discussed in considerable detail in *Data Structures and Program Design*, by R. L. Kruse, Prentice-Hall, Englewood Cliffs, N.J., 1984, pp. 112–126, and *Data Structures with Abstract Data Types and PASCAL*, by D. F. Stubbs and N. W. Webre, Brooks/Cole Publishing, Monterey, Calif., 1985, pp. 310–336.

In the prior art, such storage space contamination was avoided by deletion procedures that eliminated deleted records by replacing the deleted record with another record in the collision-resolution chain of records and thus close the chain without leaving any deleted records. One such procedure is shown in the aforementioned text by Knuth at page 527. Unfortunately, such non-contaminating procedures, due to the necessity for successive probes into the storage space, take so much time that they can be used only when the data base is off line and hence not available for accessing.

The problem, then, is to provide the speed of access of hashing techniques for large and heavily used information storage systems having expiring data and, at the same time, prevent the large-scale contamination which normally results from expired records in such large and heavily used systems.

SUMMARY OF THE INVENTION

In accordance with the illustrative embodiment of the invention, these and other problems are overcome by using a garbage collection procedure "on the fly" while other types of access to the storage space are taking place. In particular, during normal data insertion or retrieval probes into the data store, the expired, obsolete records are identified and removed in the neighborhood of the probe. Specifically, expired or obsolete records in the collision-resolution chain including the record to be accessed are removed as part of the normal retrieval procedure.

This incremental garbage collection technique has the decided advantage of automatically eliminating contamination caused by obsolete or expired records without requiring that the data base be taken off-line for such garbage collection. This is particularly important for data bases requiring rapid access and continuous availability to the user population.

BRIEF DESCRIPTION OF THE DRAWING

A complete understanding of the present invention may be gained by considering the following detailed description in conjunction with the accompanying drawing, in which:

FIG. 1 shows a general block diagram of a computer system hardware arrangement in which the information storage and retrieval system of the present invention might be implemented;

FIG. 2 shows a general block diagram of a computer system software arrangement in which the information storage and retrieval system of the present invention might find use;

FIG. 3 shows a general flow chart for table searching operation which might be used in a hashed storage system in accordance with the present invention;

FIG. 4 shows a general flow chart for a garbage collecting remove procedure which forms part of the table searching operation of FIG. 3;

FIG. 5 shows a general flow chart for record insertion operations which might be used in a hashed storage system in accordance with the present invention;

FIG. 6 shows a general flow chart for a record retrieval operation for use in a hashed storage system in accordance with the present invention; and

FIG. 7 shows a general flow chart for a record deletion operation which might be used in the hashed storage system in accordance with the present invention.

To facilitate reader understanding, identical reference numerals are used to designate elements common to the figures.

DETAILED DESCRIPTION

Referring more particularly to FIG. 1 of the drawings, there is shown a general block diagram of a computer hardware system comprising a Central Processing Unit (CPU) 10 and a Random Access Memory (RAM) unit 11. Computer programs stored in the RAM 11 are accessed by CPU 10 and executed, one instruction at a time, by CPU 10. Data, stored in other portions of RAM 11, are operated upon by the program instructions accessed by CPU 10 from RAM 11, all in accordance with well-known data processing techniques.

Central Processing Unit (CPU) 10 also controls and accesses a disk controller unit 12 which, in turn, accesses digital data stored on one or more disk storage units such as disk storage unit 13. In normal operation, programs and data are stored on disk storage unit 13 until required by CPU 10. At this time, such programs and data are retrieved from disk storage unit 13 in blocks and stored in RAM 11 for rapid access.

Central Processing Unit (CPU) 10 also controls an Input-Output (IO) controller 14 which, in turn, provides access to a plurality of input devices such as CRT (cathode ray tube) terminal 15, as well as a plurality of output devices such as printer 16. Terminal 15 provides a mechanism for a computer operator to introduce instructions and commands into the computer system of FIG. 1, and may be supplemented with other input devices such as card and tape readers, remotely located terminals, optical readers and other types of input devices. Similarly, printer 16 provides a mechanism for displaying the results of the operation of the computer system of FIG. 1 for the computer user. Printer 16 may similarly be supplemented by line printers, cathode ray tube displays, phototypesetters, graphical plotters and other types of output devices.

The constituents of the computer system of FIG. 1 and their cooperative operation are well-known in the art and are typical of all computer systems, from small personal computers to large main frame systems. The architecture and operation of such systems are well-known and, since they form no part of the present invention, will not be further described here.

In FIG. 2 there is shown a graphical representation of a typical software architecture for a computer system such as that shown in FIG. 1. The software of FIG. 2 comprises an access mechanism 20 which, for simple personal computers, may comprise no more than turning the system on. In larger systems, providing service to a larger number of users, login and password procedures would typically be implemented in access mechanism 20. Once access mechanism 20 has completed the login procedure, the user is placed in the operating system environment 21. Operating system 21 coordinates the activities of all of the hardware components of the computer system (shown in FIG. 1) and provides a number of utility programs 22 of general use to the computer user. Utilities 22 might, for example, comprise assemblers and compilers, mathematical routines, basic file handling routines and system maintenance facilities.

Many computer software systems also include a data base manager program 23 which controls access to the data records in a data base 24. Data base 24 may, for example, reside on a disk storage unit or units such as disk storage unit 13 of FIG. 1. User application programs such as application program 25 then use the data base manager program 23 to access data base records in data base 24 for adding, deleting and modifying data records. It is the efficient realization of a data base manager such as data base manager program 23 in FIG. 2 to which the present invention is directed.

Before proceeding to a description of one embodiment of the present invention, it is first useful to discuss hashing techniques in general. Hashing techniques have been used classically for very fast access to static, short term data such as a compiler symbol table. Typically, in such storage tables, deletions are infrequent and the need for the storage table disappears quickly.

In some common types of data storage systems, data records become obsolete merely by the passage of time or by the occurrence of some event. If such expired, lapsed or obsolete records are not removed from the storage table, they will, in time, seriously degrade or contaminate the performance of the retrieval system. Contamination arises because of the ever-increasing need to search longer and longer chains of record locations, many of which are expired, to reach a desired location.

More particularly, a hash table can be described as a logically contiguous, circular list of consecutively numbered, fixed-sized storage units, called cells, each capable of storing a single item called a record. Each record contains a distinguishing field, called the key, which is used as the basis for storing and retrieving the associated record. The keys throughout the hash table data base are distinct and unique for each record. Hashing functions which associate keys with storage addresses are usually not one-to-one in that they map many distinct keys into the same location.

To store a new record, a cell number is generated by invoking the hashing function on the key for the new record. If this cell location is not occupied, the new record is stored there. If this cell location is occupied, a collision has occurred and the new record must be stored elsewhere, in an overflow area, using an appropriate collision-resolution technique. A common collision-resolution strategy, which will be described here, is known as linear probing under open addressing. Open addressing means that the overflow area is the entire hash table itself. Linear probing indicates sequential scanning of cells beginning with the next cell, recalling that the storage table is viewed circularly. The collision is resolved by storing the record in the first unoccupied cell found.

To retrieve a record, the key is hashed to generate a cell location. If the record is not there (the keys do not match), searching continues following the same forward path as record storage. An empty cell terminates the retrieval procedure, which has then failed to find the record to be retrieved.

In FIG. 3 there is shown a flowchart of a *search table* procedure for searching the hash table preparatory to inserting, retrieving or deleting a record. The hash table may, for example, comprise the data base 24 of FIG. 2

and the *search table* procedure of FIG. 3 comprise a portion of the data base manager 23 of FIG. 2. Starting in box 30 of the *search table* procedure of FIG. 3, the search key of the record being searched for is hashed in box 31 to provide the address of a cell. In box 32, the empty cell just past the end of the search chain of non-empty cells is located, i.e., the first succeeding unoccupied cell is found. In box 33, the procedure moves one cell backward from the current cell position (now at the end of the chain). Decision box 34 examines the cell to determine whether the cell is empty or not. If the cell tested in decision box 34 is empty, decision box 35 is entered to determine if a key match was previously found in decision box 41 (as will be described below). If so, the search is successful and returns success in box 36 and terminates in terminal box 39. If not, box 37 is entered where the location of the empty cell is saved for possible record insertion. In box 38 failure is returned since an empty cell was found before a cell with a matching key. The procedure again terminates in box 39.

If the cell tested in decision box 34 is not empty, decision box 40 is entered to determine if the record in that cell has expired. This is determined by comparing some portion of the contents of the record to some external condition. A timestamp in the record, for example, could be compared with the time-of-day. Alternatively, the occurrence of an event can be compared with a field identifying that event in the record. In any event, if the record has not expired, decision box 41 is entered to determine if the key in this record matches the search key. If it does, the cell location is saved in box 42 and the procedure returns to box 33. If the record key does not match the search key, the procedure returns directly to box 33.

If decision box 40 determines that the record has expired, box 43 is entered to perform a non-contaminating deletion of the expired record, as will be described in connection with FIG. 4. In general, the procedure of box 43 (FIG. 4) operates to move a record further toward the end of the chain into the position of the record which has expired, thereby removing the expired record and, at the same time, closing the search chain.

It can be seen that the *search table* procedure of FIG. 3 operates to examine the entire chain of records of which the searched-for record is a part, and to delete expired records by chain-filling rather than by marking such records as deleted. In this way, contamination of the storage space by expired records is removed in the vicinity of each new table search. If contamination becomes too large even with such automatic garbage collection, then the insertion of new records can be inhibited until the *search table* procedure has had a chance to remove a sufficient number of expired records to render the operation of the system sufficiently efficient.

The *search table* procedure illustrated generally in FIG. 3 is implemented in the Appendix as PASCAL-like pseudocode. Source code suitable for compilation and execution on any standard hardware and software computing system can readily be devised from this pseudocode and the flowcharts of the figures by any person of ordinary skill in the art.

In FIG. 4 there is shown a flowchart of a *remove* procedure which removes records from the database, either records to be deleted or expired records. In general, this is accomplished by traversing the chain of the record to be removed in a forward direction searching for a record whose key hashes at or behind the cell to be removed. When such a record is found, it is copied to the cell of the record to be removed. The copied record is then taken as the record to be removed and the process is continued until the end of the search chain is reached. In box 54, the final copied record is marked empty prior to teminating the procedure. The *remove* procedure of FIG. 4 might comprise a portion of the data base manager program 23 of FIG. 2.

Starting at starting box 50 of FIG. 4, the procedure is entered with the location of a cell to be removed which is called the base cell. Initially, box 51 is entered where the load count in the table is adjusted to reflect the removal of one record. The load, of course, is the number of occupied cells. As previously noted, the value of this load can be used to disable the insertion of new records until the load has reached a low enough value to permit efficient searching. In box 52, the procedure of FIG. 4 advances to the next cell in the chain beyond the base cell. In decision box 53 this cell is tested to see if it is empty. If it is empty, the end of the chain has been reached and box 54 is entered to mark the base cell as empty. Decision box 55 is then entered to determine if a record was found (by the *search table* procedure) which matched the search key and, if so, the procedure is terminated in terminal box 56. If a matching record was not found, decision box 57 is entered to determine if the base cell is ahead of the hash location of the search key. If not, the procedure is terminated in box 56. If the base cell does hash ahead of the search record, then the base cell can be used for storing a new record. In box 58, the location of this empty cell is therefore saved as a possible insertion site.

Returning to box 53, if the next cell is not empty, box 59 is entered to determine if the record in this cell hashes ahead of the base cell. If so, box 52 is re-entered to advance to the next cell in the chain. If this next cell hashes at or behind the base cell, however, box 60 is entered to copy the contents of this next cell to the base cell, thereby obliterating (removing) the base cell contents. Box 61 is then entered to test if the *search table* procedure found a matching record. If not, box 52 is re-entered to advance to the next cell. If a matching record was found, decision box 62 is entered to test if the matching record is the base cell record. If not, box 52 is re-entered to advance to the next cell. If the matching record is the base cell, however, box 63 is entered to store the location of the former base cell as the position of the matching record and then box 52 is re-entered to advance to the next cell in the search chain.

It can be seen that the procedure of FIG. 4 operates to examine the entire search chain and to move records from later positions in the chain to vacated positions in the chain such that the chain is entirely closed at the end of the procedure. That is, no empty cells are left to erroneously break up a search chain. As noted in connection with FIG. 3, expired records are subjected to the *remove* procedure of FIG. 4. As will be noted in connection with FIG. 7, records to be deleted from the data base are also subjected to the *remove* procedure of FIG. 4.

The *remove* procedure illustrated generally in FIG. 4 is implemented in the Appendix as PASCAL-like pseudocode. Source code suitable for compilation and execution on any standard hardware and software computing system can readily be devised from this pseudo-

code and the flowchart of FIG. 4 by any person of ordinary skill in the art.

In FIG. 5 there is shown a detailed flowchart of an *insert* procedure suitable for use in the information storage and retrieval system of the present invention. The *insert* procedure of FIG. 5 begins as starting box 70 from which box 71 is entered. In box 71, the *search table* procedure of FIG. 3 is invoked with the search key of the record to be inserted. As noted in connection with FIG. 3, the *search table* procedure locates the target cell location and, if part of a search chain, removes all expired cells from that search chain. Decision box 72 is then entered where it is determined whether or not the *search table* procedure found a record with a matching key. If so, box 73 is entered where the record to be inserted in put into the storage table in the position of the old record with a matching key. In box 74, the *insert* procedure reports that the old record has been replaced by the new record and the procedure is terminated in terminal box 75.

Returning to decision box 72, if a matching record is not found, decision box 76 is entered to determine if the table load is below a preselected threshold (typically about 75% of the table capacity). If the load is not below the threshold, the storage table is too full to be access efficiently, and box 77 is entered to report that the the table is full and the record cannot be inserted. The procedure then terminates in terminal box 75. If the load is below the threshold, box 78 is entered where the record to be inserted is placed in the empty cell position found by the *search table* procedure. In box 79, the load is adjusted to reflect the addition of one record to the storage table, the procedure reports that the record was inserted in box 80 and the procedure terminated in box 75.

The *insert* procedure illustrated generally in FIG. 5 is implemented in the Appendix as PASCAL-like pseudocode. Source code suitable for compilation and execution on any standard hardware and software computing system can readily be devised from this pseudocode and the flowcharts of the FIG. 5 by any person of ordinary skill in the art.

In FIG. 6 there is show a detailed flowchart of a *retrieve* procedure which is used to retrieve a record from the data base 24 of FIG. 2. Starting in box 90, the *search table* procedure is invoked in box 91, using the key of the record to be retrieved as the search key. In box 92 it is determined if a record with a matching key was found by the *search table* procedure. If not, box 93 is entered to report failure of the *retrieve* procedure and the procedure is terminated in box 96. If a matching record was found, box 94 is entered to copy the matching record into a buffer store for processing by the calling program, box 95 is entered to return an indication of successful retrieval and the procedure terminated in box 96.

The pseudo-code for the *retrieve* procedure of FIG. 6 is included in the Appendix. Executable code for all common hardware and system software arrangements can readily be devised by those skilled in the art from the flowchart and the pseudo-code.

In FIG. 7 there is shown a detailed flowchart of a *delete* procedure useful for actively removing records from the data base 24 of FIG. 2. Starting at box 100, the procedure of FIG. 7 first invokes the *search table* procedure of FIG. 3 in box 101, using the key of the record to be deleted as the search key. In box 102, it is determined if the *search table* procedure was able to locate a record with a matching key. If not, box 103 is entered to report failure of the deletion procedure and the procedure is terminated in box 106. If a matching record was found, as determined by box 102, the *remove* procedure of FIG. 4 is invoked in box 104. As noted in connection with FIG. 4, this procedure removes the record to be deleted and, at the same time, closes the search chain. Box 105 is then entered to report successful deletion to the calling program and the procedure is terminated in box 106.

The *delete* procedure illustrated generally in FIG. 7 is implemented in the Appendix as PASCAL-like pseudocode. Source code suitable for compilation and execution on any standard hardware and software computing system can readily be devised from this pseudocode and the flowchart of FIG. 7 by any person of ordinary skill in the art.

The attached Appendix contains pseudocode listings for all of the programmed functions necessary to implement a data base manager 23 (FIG. 2) operating in accordance with the present invention. These listings follow the flowcharts of FIGS. 3–7 and further explain and elucidate the flowcharts. Any person of ordinary skill in the art will have no difficulty implementing these functions in any desired program language to run on any desired computer hardware configuration.

It should also be clear to those skilled in the art that further embodiments of the present invention may be made by those skilled in the art without departing from the teachings of the present invention.

APPENDIX

```
Functions Provided

The following functions are made available to the
application program:

insert (record: record type)

Returns replaced if a record associated with record.key
was found in the table and subsequently replaced.

Returns inserted if a record associated with record.key
was not found in the table and the passed record was
subsequently inserted.

Returns full if a record associated with record.key was
not found in the table and passed record could not be
inserted because load factor has reached max load factor.

retrieve (record: record type)

Returns success if record associated with record.key was
found in the table and assigned to record.

Returns failure if search was unsuccessful.

delete (record key: record key type)

Returns success if record associated with record key was
found in the table and subsequently deleted

Returns failure if none found.

Definitions

The following formal definitions are required for
specifying the insertion, retrieval, and deletion
algorithms:

const table size          /* size of hash table */
```

```
const max load factor     /* 0 ≤ max load factor < 1 */

var table: array[0 .. table size-1] of record type;
                /* hash table */

var load: 0 .. table size-1;
                /* number of occupied entries of
                   hash table array (initially 0) */


Algorithms

Algorithms for the functions described above are
given below:
function insert (record: record type):
        (replaced, inserted, full );

  var position: 0 .. table size-1;
                /* position in table to update or
                   insert (returned by search table) */

  begin

    if search table (record.key, position)

       then begin

         table[position] := record;
         return (replaced)

       end

       else if load/table size < max load factor

         then begin

           load := load+1;
           table[position] := record;
           return (inserted)

         end

         else return (full)

  end                /* insert */


  function retrieve (var record: record type):
          (success, failure);

    var position: 0 .. table size-1;
                /* position in table where record
                   resides (returned by search table) */

    begin

      if search table (record.key, position)

         then begin

           record := table[position];
           return (success)

         end

         else return (failure)

    end  /* retrieve */

  function delete (record key: record key type):
         (success, failure);

    var position: 0 .. table size-1;
                /* position in table where record
                   resides (returned by search table) */
    dummy variable: 0 .. table size-1;
                /* last two arguments to remove are
                   not relevant here                 */


    begin

      if search table (record key, position)

         then begin

           remove (position, true, dummy variable,
                   dummy variable);

           return (success)

         end

         else return (failure)

    end    /* delete */


  function search table (record key: record key type;
          var position: 0 .. table size-1): boolean;

  /* search table for record key and delete expired
     expired records in target chain: position is set to
     index of found record or appropriate empty cell */

    var i: 0 .. table size-1;
          /* used for scanning chain,
             both forwards & backwards */

        pos empty: 0 .. table size-1;
          /* index of leftmost empty cell
             to right of position */

        is rec found: boolean;
          /* indicates whether search is successful */


  begin

    position := hash (record key);
    is rec found :=  false;

    if table[position] is not empty then


      begin
        i := position; /* loop initialization */
        repeat         /* scan forward to end of chain
                          containing table[position]  */

            i := (i+1) mod table size

        until (table[i] is empty);

        pos empty := i;
      i := (i-1+table size) mod table size;

      while (table[i] is not empty) do
            /*scan chain in reverse,
              deleting expired entries */

        begin

          if table[i] is expired
                    then remove (i, is rec found,
                                 position, pos empty)

                    else if table[i].key = record key
```

```
                then begin

                    is rec found := true;
                    position := i

                end;

            i := (i-1+table size) mod table size

        end;   /* while */

    if not is rec found then position := pos empty

    end;    /* then  */

  return (is rec found)

end                    /* search table  */

procedure remove (cell to del:
 0 .. table size-1; is rec found: boolean;
 var pos of search rec, pos empty: 0 .. table size-1;

          /* Delete table[cell to del] */

var i, j: 0 .. table size-1;

begin

   load := load-1;

   do forever

     i := cell to del;
             /* save position of emptied slot */

     repeat          /* scan forward looking for a
                        record to fill hole in chain */

        cell to del := (cell to del+1) mod table size;

        if table[cell to del] is empty

          then begin

             table[i] := empty;

             if not is rec found then

                if (pos of search rec ≤ i < pos empty)
                or (i< pos empty < pos of search rec)
                or (pos empty< pos of search rec ≤
                i) then pos empty := i;

             return
            end;

        j := hash (table[cell to del].key)

     until (j ≤ i < cell to del)
     or (i < cell to del < j)
     or (cell to del< j ≤ i);

     table[i] := table[cell to del];
     /* use table[cell to del] to plug hole in chain */

     if (is rec found) and
     (pos of search rec = cell to del)
     then pos of search rec := i

   end

end                    /* remove */
```

What is claimed is:

1. An information storage and retrieval system using hashing techniques to provide rapid access to the records of said system and utilizing a linear probing technique to store records with the same hash address, at least some of said records automatically expiring, said system comprising

a record search means utilizing a search key to access a chain of records having the same hash address,

said record search means including means for identifying and removing all expired ones of said records from said chain of records each time said chain is accessed, and

means, utilizing said record search means, for inserting retrieving and deleting records from said system and, at the same time, removing all expired ones of said records in the accessed chains of records.

2. The information storage and retrieval system according to claim 1 further comprising

means for recursively moving a record from a later position in said chain of records into the position of one of said expired records.

3. The information storage and retrieval system according to claim 1 further including

means for counting the number of records in said system,

means, responsive to said counting means, for inhibiting the insertion of new records into said system when the number of records in said system exceeds a preselected value.

4. The information storage and retrieval system according to claim 3 further including

means, also responsive to said counting means, for re-enabling the insertion of new records into said system when the number of records in said system falls below said preselected value.

5. A method for storing and retrieving information records using hashing techniques to provide rapid access to said records and utilizing a linear probing technique to store records with the same hash address, at least some of said records automatically expiring, said method comprising the steps of

accessing a chain or records having the same hash address,

identifying the automatically expired ones of said records,

removing all automatically expired records from said chain of records each time said chain is accessed, and

inserting, retrieving or deleting one of said records from said system following said step of removing.

6. The method according to claim 5 further comprising the step of

moving a record from a later position in said chain of records into the position of one of said expired records.

7. The method according to claim 5 further comprising the steps of

counting the number of records in said system, and

inhibiting the insertion of new records into said system when the number of records in said system rises above a preselected value.

8. The method according to claim 7 further comprising the step of

re-enabling the insertion of new records into said system when the number of records in said system falls below said preselected value.

* * * * *

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Attorney Docket No.: M32-001

Applicant : JAE-KYOON KIM, JIN-HAK LEE, KWANG-HOON PARK, JOO-HEE MOON & SUNG-MOON CHUN
Serial No : 08/740,300
Filed : 10/25/96
For : METHOD OF GENERATING OBJECT-ADAPTIVE CODE BLOCK PATTERN
Art Unit : 2608

Assistant Commissioner for Patents
Washington, D.C. 20231

STATUS REQUEST

Sir:

Please advise the status of the above-identified application and when we can expect to receive an official communication.

Respectfully Submitted,

Henry D. Coleman

McAULAY FISHER NISSEN GOLDBERG & KIEL, LLP

HENRY D. COLEMAN
REG. No. 32,559

261 Madison Avenue
New York, New York 10016
(212) 986-4090

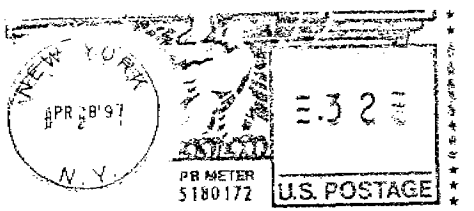
Dated : 04/28/97

CERTIFICATE OF MAILING

I hereby certify that this paper and every paper referred to therein as being enclosed is being deposited with the U.S. Postal Service as first class mail, postage prepaid, in an envelope addressed to :
ASST. Commissioner of Patents, Washington, DC 20231,
on 04/28/97 (Date of Deposit)

04/28/97 Date RUTH MONTALVO

McAULAY FISHER NISSEN GOLDBERG & KIEL, LLP
261 MADISON AVENUE, NEW YORK, N.Y. 10016-2391



POSTAGE PREPAID

McAULAY FISHER NISSEN GOLDBERG & KIEL, LLP
261 Madison Avenue
New York, NY 10016-2391


PATENT ROOM
MAY
1997
TRADEMARK OFF.

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.
08/775,864	01/02/97	NEMES	R

LM41/0420

RICHARD MICHAEL NEMES
1432 EAST 35TH STREET
BROOKLYN NY 11234-2604

EXAMINER	
ALAM, H	
ART UNIT	**PAPER NUMBER**
2771	3

DATE MAILED: 04/20/98

Please find below and/or attached an Office communication concerning this application or proceeding.

Commissioner of Patents and Trademarks

Please see attachment

PTO-90C (Rev. 2/95)
*U.S. GPO: 1997-422-196/60031

1- File Copy

Office Action Summary

Application No.	Applicant(s)
08/775,864	Nemes
Examiner	**Group Art Unit**
Hosain T. Alam	2771

☐ Responsive to communication(s) filed on _____.

☐ This action is **FINAL**.

☐ Since this application is in condition for allowance except for formal matters, **prosecution as to the merits is closed** in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11; 453 O.G. 213.

A shortened statutory period for response to this action is set to expire 3 month(s), or thirty days, whichever is longer, from the mailing date of this communication. Failure to respond within the period for response will cause the application to become abandoned. (35 U.S.C. § 133). Extensions of time may be obtained under the provisions of 37 CFR 1.136(a).

Disposition of Claims

☒ Claim(s) *1-8* is/are pending in the application.

Of the above, claim(s) _____ is/are withdrawn from consideration.

☐ Claim(s) _____ is/are allowed.

☒ Claim(s) *1-8* is/are rejected.

☐ Claim(s) _____ is/are objected to.

☐ Claims _____ are subject to restriction or election requirement.

Application Papers

☒ See the attached Notice of Draftsperson's Patent Drawing Review, PTO-948.

☐ The drawing(s) filed on _____ is/are objected to by the Examiner.

☐ The proposed drawing correction, filed on _____ is ☐approved ☐disapproved.

☐ The specification is objected to by the Examiner.

☐ The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. § 119

☐ Acknowledgement is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d).

☐ All ☐ Some* ☐ None of the CERTIFIED copies of the priority documents have been

☐ received.

☐ received in Application No. (Series Code/Serial Number) _____.

☐ received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

*Certified copies not received: _____.

☐ Acknowledgement is made of a claim for domestic priority under 35 U.S.C. § 119(e).

Attachment(s)

☒ Notice of References Cited, PTO-892

☒ Information Disclosure Statement(s), PTO-1449, Paper No(s). 2

☐ Interview Summary, PTO-413

☒ Notice of Draftsperson's Patent Drawing Review, PTO-948

☐ Notice of Informal Patent Application, PTO-152

--- SEE OFFICE ACTION ON THE FOLLOWING PAGES ---

Part III DETAILED ACTION

1. Claims 1-8 are pending in this application.

Drawings

2. This application has been filed with informal drawings which are acceptable for examination purposes only. Formal drawings will be required when the application is allowed.

Double Patenting

3. The nonstatutory double patenting rejection is based on a judicially created doctrine grounded in public policy (a policy reflected in the statute) so as to prevent the unjustified or improper timewise extension of the "right to exclude" granted by a patent and to prevent possible harassment by multiple assignees. See *In re Goodman*, 11 F.3d 1046, 29 USPQ2d 2010 (Fed. Cir. 1993); *In re Longi*, 759 F.2d 887, 225 USPQ 645 (Fed. Cir. 1985); *In re Van Ornum*, 686 F.2d 937, 214 USPQ 761 (CCPA 1982); *In re Vogel*, 422 F.2d 438, 164 USPQ 619 (CCPA 1970);and, *In re Thorington*, 418 F.2d 528, 163 USPQ 644 (CCPA 1969).

A timely filed terminal disclaimer in compliance with 37 CFR 1.321© may be used to overcome an actual or provisional rejection based on a nonstatutory double patenting ground provided the conflicting application or patent is shown to be commonly owned with this application. See 37 CFR 1.130(b).

Effective January 1, 1994, a registered attorney or agent of record may sign a terminal disclaimer. A terminal disclaimer signed by the assignee must fully comply with 37 CFR 3.73(b).

4. Claims 1-4 are rejected under the judicially created doctrine of double patenting over claim 1 of U. S. Patent No. 5,121,495 issued to Nemes, hereinafter ('495) since

the claims, if allowed, would improperly extend the "right to exclude" already granted in the patent.

5. The subject matter claimed in the instant application is fully disclosed in the patent and is covered by the patent since the patent and the application are claiming common subject matter, as follows:

The retrieval system as claimed in Claim 1 of the '495 patent is directed to an apparatus and method for information storage and retrieval wherein the memory addresses are hashed by using a chain of records having same hash address (see claim 1, col. 12, line 7-8) and wherein the step of removing the expired records is also included.

The "chain of records" is equivalent to a linked list of pointers/addresses of records as claimed and the "chaining" is equivalent to being linked.

As to claims 1 and 3: The '495 patent does not recite the term, "linked list" and instead recites "chain of records".

It would have been obvious to a person of ordinary skill in the art at the time the invention was made to use a linked list of records because a chain of records generates a linked list.

As to claims 2 and 4: The '495 patent does not recite the removal based on the determination of a maximum number of expired records.

It would have been obvious to a person of ordinary skill in the art at the time the invention was made to group a number of records and thus to predetermine the maximum number in the group to facilitate an efficient processing of records and to reduce the system overhead by avoiding expensive I/O operations based on the number of records in the grouping. The number of records in a group determines the bytes of memory required in the group.

6. Claims 5-8 are rejected under the judicially created doctrine of double patenting over claims 1 and 2 of U. S. Patent No. 5,287,499 issued to Nemes, hereinafter ('499) since the claims, if allowed, would improperly extend the "right to exclude" already granted in the patent.

The subject matter claimed in the instant application is fully disclosed in the patent and is covered by the patent since the patent and the application are claiming common subject matter, as follows:

The retrieval system as claimed in Claims 1 and 2 of the '499 patent is directed to an apparatus and method for information storage and retrieval wherein the memory addresses are hashed by using a chain of records having same hash address, the chaining of records is external (see claim 1, col. 17, line 1).

The "external chaining of records" is equivalent to a linked list of pointers/addresses of records as claimed and the "chaining" is equivalent to being linked.

As to claims 5 and 7: The '499 patent does not recite the term, "linked list" and instead recites "external chaining" and further does not recite the terms, "insert, retrieve or delete" instead recites "storing" (see claim 2, col. 18).

It would have been obvious to a person of ordinary skill in the art at the time the invention was made to use a linked list of records because a chain of records chained by an external chaining generates a linked list.

As to claims 7 and 8: The '499 patent does not recite a "maximum number of records" instead recite a "threshold".

It would have been obvious to a person of ordinary skill in the art at the time the invention was made to group a number of records for determining the threshold and thus to predetermine the maximum number for the threshold to facilitate an efficient processing of records and to reduce the system overhead by avoiding expensive I/O operations based on the number of records associated with the threshold. The number of records associated with the threshold determines the bytes of memory required for the threshold.

7. Furthermore, there is no apparent reason why applicant was prevented from presenting claims corresponding to those of the instant application during prosecution of the application which matured into a patent. See *In re Schneller*, 397 F.2d 350, 158 USPQ 210 (CCPA 1968). See also MPEP § 804.

Claim Rejections - 35 USC § 103

8. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

9. Claims 1-8 are rejected under 35 U.S.C. § 103 as being unpatentable over U.S. Patent No. 5,287,499 issued to Nemes ('499) in view of U.S. Patent No. 5,202,981 issued to Shackelford ("Schackelford").

10. With respect to claims 1-8, Nemes teaches everything that is claimed (col. 2, line 60-64, col. 6, line 49-51; '499 reference) except that it does not explicitly indicate the determination of threshold as being the maximum number of records.

Schackelford teaches the maximum number of pointers to records wherein the records are linked in a bidirectional linked list (col. 3, line 61 through col. 4, line 2).

It would have been obvious to a person of ordinary skill in the art at the time the invention was made to determine the maximum number of records to make the best use of the memory space available to a user and thus to reduce the system overhead (col. 4, lines 1-2, Schackelford).

11. The above reasoning does not discuss the linked list and the step of removing the records, it rather addresses the issue of determining the maximum number of

records. For the limitations directed to the linked lists and the step of removing, the Applicant is requested to refer to the Obvious Double Patenting rejection set forth hereinabove where they have been discussed in details.

Contact Information

12. Direct inquiries concerning this communication should be directed to Hosain Alam whose telephone number is (703) 308-6662. The examiner can normally be reached on Monday - Thursday from 8:00 AM to 4:30 PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Thomas Black, can be reached on (703)305-9707.

Any response to this action should be mailed to:

Commissioner of Patents and Trademarks

Washington, D.C. 20231

or faxed to:

(703) 308-9051, (for formal communications intended for entry)

Or:

(703) 305-9724 or (703) 308-6606 (for informal or draft communications, please label "PROPOSED" or "DRAFT")

Hand-delivered responses should be brought to Crystal Park II, 2121 Crystal Drive, Arlington. VA., Sixth Floor (Receptionist).

Inquiries of a general nature or relating to the status of this application should be directed to the Group receptionist whose telephone number is (703) 305-9600.

H.A.

April 14, 1998

THOMAS G. BLACK
SUPERVISORY PATENT EXAMINER
GROUP 2700

Form PTO 948 (Rev. 10-94) U.S. DEPARTMENT OF COMMERCE - Patent and Trademark Office Application No. 775864

NOTICE OF DRAFTSPERSON'S PATENT DRAWING REVIEW

PTO Draftpersons review all originally filed drawings regardless of whether they are designated as formal or informal. Additionally, patent Examiners will review the drawings for compliance with the regulations. Direct telephone inquiries concerning this review to the Drawing Review Branch, 703-305-8404.

The drawings filed (insert date) 1/2/97 are
A ___ not objected to by the Draftsperson under 37 CFR 1.84 or 1.152.
B ___ objected to by the Draftsperson under 37 CFR 1.84 or 1.152 as indicated below. The Examiner will require submission of new, corrected drawings when necessary. Corrected drawings must be submitted according to the instructions on the back of this Notice.

1. DRAWINGS. 37 CFR 1.84(a): Acceptable categories of drawings: Black ink. Color.
   - ___ Not black solid lines. Fig(s)___
   - ___ Color drawings are not acceptable until petition is granted. Fig(s) ___
2. PHOTOGRAPHS. 37 CFR 1.84(b)
   - ___ Photographs are not acceptable until petition is granted. Fig(s) ___
   - ___ Photographs not properly mounted (must use brystol board or photographic double-weight paper). Fig(s)___
   - ___ Poor quality (half-tone). Fig(s)___
3. GRAPHIC FORMS. 37 CFR 1.84 (d)
   - ___ Chemical or mathematical formula not labeled as separate figure. Fig(s)___
   - ___ Group of waveforms not presented as a single figure, using common vertical axis with time extending along horizontal axis. Fig(s)___
   - ___ Individuals waveform not identified with a separate letter designation adjacent to the vertical axis. Fig(s)___
4. TYPE OF PAPER. 37 CFR 1.84(c)
   - ___ Paper not flexible, strong, white, smooth, nonshiny, and durable. Sheet(s)___
   - ___ Erasures, alterations, overwritings, interlineations, cracks, creases, and folds copy machine marks not accepted. Fig(s) 1-7
   - ___ Mylar, velum paper is not acceptable (too thin). Fig(s) ___
5. SIZE OF PAPER. 37 CFR 1.84(f): Acceptable sizes:
   - 21.6 cm. by 35.6 cm. (8 1/2 by 14 inches)
   - 21.6 cm. by 33.1 cm. (8 1/2 by 13 inches)
   - 21.6 cm. by 27.9 cm. (8 1/2 by 11 inches)
   - 21.0 cm. by 29.7 cm. (DIN size A4)
   - ___ All drawing sheets not the same size. Sheet(s)___
   - ___ Drawing sheet not an acceptable size. Sheet(s) ___
6. MARGINS. 37 CFR 1.84(g): Acceptable margins:

Paper size			
21.6 cm. X 35.6 cm. (8 1/2 X 14 inches)	21.6 cm X 33.1 cm. (8 1/2 X 13 inches)	21.6 cm. X 27.9 cm. (8 1/2 X 11 inches)	21.0 cm. X 29.7 cm. (DIN Size A4)
T 5.1 cm. (2")	2.5 cm. (1")	2.5 cm. (1")	2.5cm.
L .64 cm. (1/4")	.64 cm. (1/4")	.64 cm. (1/4")	2.5 cm.
R .64 cm. (1/4")	.64 cm. (1/4")	.64 cm. (1/4")	1.5 cm.
B .64 cm. (1/4")	.64 cm. (1/4")	.64 cm. (1/4")	1.0 cm.

Margins do not conform to chart above.
Sheet(s)___
___Top (T) ___ Left (L) ___Right (R) ___Bottom (B)

7. VIEWS. 37 CFR 1.84(h)
   REMINDER: Specification may require revision to correspond to drawing changes.
   - ___ All views not grouped together. Fig(s)___
   - ___ Views connected by projection lines or lead lines. Fig(s)___
   - Partial views. 37 CFR 1.84(h) 2
   - ___ View and enlarged view not labled separatly or properly. Fig(s)___
   - Sectional views. 37 CFR 1.84 (h) 3
   - ___ Hatching not indicated for sectional portions of an object. Fig(s)___
   - ___ Cross section not drawn same as view with parts in cross section with regularly spaced parallel oblique strokes. Fig(s)___
8. ARRANGEMENT OF VIEWS. 37 CFR 1.84(i)
   - ___ Words do not appear on a horizontal, left-to-right fashion when page is either upright or turned so that the top becomes the right side, except for graphs. Fig(s)___
9. SCALE. 37 CFR 1.84(k)
   - ___ Scale not large enough to show mechanism with crowding when drawing is reduced in size to two-thirds in reproduction. Fig(s)___
   - ___ Indication such as "actual size" or scale 1/2" not permitted. Fig(s)___
10. CHARACTER OF LINES, NUMBERS, & LETTERS. 37 CFR 1.84(l)
    - ___ Lines, numbers & letters not uniformly thick and well defined, clean, durable and black (except for color drawings). Fig(s) 1-7
11. SHADING. 37 CFR 1.84(m)
    - ___ Solid black shading areas not permitted. Fig(s)___
    - ___ Shade lines, pale, rough and blurred. Fig(s)___
12. NUMBERS, LETTERS, & REFERENCE CHARACTERS. 37 CFR 1.84(p)
    - ___ Numbers and reference characters not plain and legible. 37 CFR 1.84(p)(1) Fig(s) 1-7
    - ___ Numbers and reference characters not oriented in same direction as the view. 37 CFR 1.84(p)(1) Fig(s)___
    - English alphabet not used. 37 CFR 1.84(p)(2) Fig(s)___
    - ___ Numbers, letters, and reference characters do not measure at least .32 cm. (1/8 inch) in height. 37 CFR(p)(3) Fig(s) 2-4
13. LEAD LINES. 37 CFR 1.84(q)
    - ___ Lead lines cross each other. Fig(s)___
    - ___ Lead lines missing. Fig(s)___
14. NUMBERING OF SHEETS OF DRAWINGS. 37 CFR 1.84(t)
    - ___ Sheets not numbered consecutively, and in Arabic numerals, beginning with number 1. Sheet(s)___
15. NUMBER OF VIEWS. 37 CFR 1.84(u)
    - ___ Views not numbered consecutively, and in Arabic numerals, beginning with number 1. Fig(s)___
    - ___ View numbers not preceded by the abbreviation Fig. Fig(s)___
16. CORRECTIONS. 37 CFR 1.84(w)
    - ___ Corrections not made from prior PTO-948. Fig(s)___
17. DESIGN DRAWING. 37 CFR 1.152
    - ___ Surface shading shown not appropriate. Fig(s)___
    - ___ Solid black shading not used for color contrast. Fig(s)___

COMMENTS: - Change all 7 to 2
- Character must not be cut thru by structured lines
Fig. 7

ATTACHMENT TO PAPER NO. 3 REVIEWER [signature] DATE 3/11/97

PTO Copy

BTEX0000348

REMINDER

Drawing changes may also require changes in the specification, e.g., if Fig. 1 is changed to Fig. 1A, Fig. 1B, Fig. 1C, etc., the specification, at the Brief Description of the Drawings, must likewise be changed. Please make such changes by 37 CFR 1.312 Amendment at the time of submitting drawing changes.

INFORMATION ON HOW TO EFFECT DRAWING CHANGES

1. Correction of Informalities--37 CFR 1.85

File new drawings with the changes incorporated therein. The application number or the title of the invention, inventor's name, docket number (if any), and the name and telephone number of a person to call if the Office is unable to match the drawings to the proper application, should be placed on the back of each sheet of drawings in accordance with 37 CFR 1.84(c). Applicant may delay filing of the new drawings until receipt of the Notice of Allowability (PTOL-37). Extensions of time may be obtained under the provisions of 37 CFR1.136. The drawing should be filed as a separate paper with a transmittal letter addressed to the Drawing Review Branch.

2. Timing of Corrections

Applicant is required to submit **acceptable** corrected drawings within the three-month shortened statutory period set in the Notice of Allowability (PTOL-37). If a correction is determined to be unacceptable by the Office, applicant must arrange to have acceptable correction resubmitted within the original three-month period to avoid the necessity of obtaining as extension of time and paying the extension fee. Therefore, applicant should file corrected drawings as soon as possible.

Failure to take corrective action within set (or extended) period will result in **ABANDONMENT** of the Application.

3. Corrections other than Informalities Noted by the Drawing Review Branch on the Form PTO 948

All changes to the drawings, other than informalities noted by the Drawing Review Branch, **MUST** be approved by the examiner before the application will be allowed. No changes will be permitted to be made, other than correction of informalities, unless the examiner has approved the proposed changes.