A WORKSTATION ENVIRONMENT FOR RETRIEVING TEXT FROM SMALL TO LARGE UNSTRUCTURED INFORMATION RESERVOIRS

James L. Clark
Research and Development Manager
Computer Systems Advisers (Pte) Limited
Singapore

Jeffrey L. Cooke
Senior Lecturer
Capricornia Institute of Advanced Education
Queensland, Australia

ABSTRACT

Commercially available database management systems provide access to numerical data as their main forte. P-trees facilitate access to nonnumerical data; for example, free text information composed of a mixture of numerical and bibliographic data. This kind of processing requires significant string manipulation which is not available in commercial database products which operate on small workstations connected to larger hosts.

This paper describes a near optimal method of storing, searching and retrieving derived from the "trie" structure; particularly the use of tree data structures of type P, sometimes called the PATRICIA type. The method is very efficient and is suitable for use on small workstations as well as large computer systems. The development and recent applications of the paged P-tree method are described along with appropriate performance comparisons. The database applications software, based on P-trees, for providing small workstation users access and manipulation capabilities within a large information repository composed of textual and numerical data, is reviewed.

AN ENVIRONMENT FOR RETRIEVING TEXT

INTRODUCTION

The class of indexing methods based on the Fredkin "trie" structure have been known for many years, Fredkin (1960). The term "trie" is derived from letters imbedded in the word "retrieval". This data structure can be used to represent sets of character strings and/or abstract data types. Consequently, it is an extremely versatile data structure. An algorithm based on the binary trie structure was developed in the mid sixties, Morrison (1967, 1968). The method is called "Practical Algorithm To Retrieve Information Coded in Alphanumeric" or PATRICIA for brevity. The tree described by PATRICIA is called a P-tree or a tree of type PATRICIA, CLARK (1973), Lagana et al (1980).

Trie structures are closely associated with a number system. The binatural number system, B is a natural extension of the counting or Cardinal number system, Morrison (1972). B is characterised by a first member 1, and a pair of successors functions, L for left and R for right successors. B has many representations which differ not only in form but also in use. Binary P-trees are a special case of the free monoid with two generators, on the set {0, 1}, which can be formed from the Binatural number system, and plays an essential role in the coding of information for binary digital computers.

P-trees facilitate the indexing, modifying, querying and retrieving of information in a binary coded data base. The data structure is especially suitable for dealing with variable length keys, titles, and/or similar textual phrases stored within a large file, Clark (1972), Knuth (1973). The selection of a proper alphabet on which P-trees should operate need not be restricted to binary. M-ary alphabets could be used; however the allocation of storage and efficiency of inner loops are made easier if the alphabet is binary, Clark (1972). In addition the vast majority of computers manipulate and store information in a binary manner.

An example where text is an important parameter appears in enabling scientific notebooks to become machine readable. Many other examples having a commercial as well as scientific orientation are frequently encountered. Text when composed of numerical and bibliographic data provides unique and difficult challenges.

AN ENVIRONMENT FOR RETRIEVING TEXT

Very few commercially available database management systems are designed to manipulate free text information, not even those based on the relational model, Pollard et al (1984).

The manipulation of free text is left to packages such as IBM´s STAIRS, from the 1973 technological era. Products of this ilk are generally precluded for use on small workstations and even minicomputers.

A few authors briefly mention the area of text searches. Almost all available DBMS implementations and text books assume the database is "formatted" and is highly structured. Most information retrieval applications are based on unstructured textual databases, e.g. scientific abstracts and certain commercial databases involving statistical data encountered in the banking industry. Adhoc queries against these kinds of database are very complex, Date (1983).

P-trees are an excellent mechanism for providing free text manipulation not only on large computers but also on small workstations.

P-TREES OF TYPE PATRICIA-I

The PATRICIA algorithm forms an N-node binary search tree based on the binary representation of keys, without storing keys in the nodes. In particular, the index which a P-tree defines includes no keys or text, only numbers which point to other P-tree nodes or storage locations of keyed targets in the data base. The fundamental idea behind P-trees is to build a Fredkin like binary trie by avoiding one-way branches. This is done by including in each node the number of bits to skip over before making the next branch test, Knuth (1973). The contents of a P-tree index are pointers to locations within a data base target, and pointers to enable branching decisions. The method examines the bit pattern of the specified encoding, e.g. ASCII or EBCDIC. The algorithm recognises those bit locations which are determined to be important in distinguishing keys. Such bits are called "twin bits".

Consider a string of bits, i.e. a string of 0´s and 1´s, which is the encoded representation of key K. Then K has two daughters or permutations which are extensions of K; namely K0 and K1, (K with "0" or "1" concatenated). If both K0 and K1 are keys, then K is a branch key and K0 and K1 are twin keys. If only one of K0 and K1 is a key then K is not a branch key. Keys are in the same chain of keys if they have right extensions in common; hence are possible keys in one or more data base records. On a search if a key is not found, the algorithm at least locates those keys which have the most initial bits in common, Clark (1972). For a pro forma of P-trees of type PATRICIA-I, see Figure 1.

In all cases data base target records are retrieved in collating sequence order. In fact the algorithm can be applied to problems encountered in sorting and is particularly useful when sorting records involving long variable length keys in very long records, Cooke (1985).

When keys are presented to the system the computation time required to determine the presence or absence of an owner record is bounded, the bound depending linearly on the length of the key and being independent of the size of the data base and the number of occurrences. The algorithms for updating the data base are straight forward and economical in computation time, Clark (1972), Lagana (1980). As mentioned above, the index, i.e. the P-tree nodes consist only of numbers; hence the keys appear in their proper place, in their natural form in the data base. The data base is also unrestricted in format. Items in the data base do not need to be arranged in any special order. New additions do not necessitate relocating or shuffling information around in the data base. Keys and data base targets need not be in one-to-one correspondence since one data base target may be retrievable by as many keys as have been connected to it. One key may retrieve as many targets in the data base as it has occurrences.

P-TREES OF TYPE PATRICIA-II

Computer implementation of the original algorithm, PATRICIA-I, was limited because of computer memory constraints in that the P-tree had to be entirely core resident, Clark (1972). This limitation restricted the size of data bases which could be indexed as well as the operational environment. A substantial quantity of memory was required when applying the method to large volumes of data. At that time computers were not equipped with large quantities of random access memory. In fact, PATRICIA-I could efficiently handle a data base with approximately R/5 keys, R being the number of random memory words

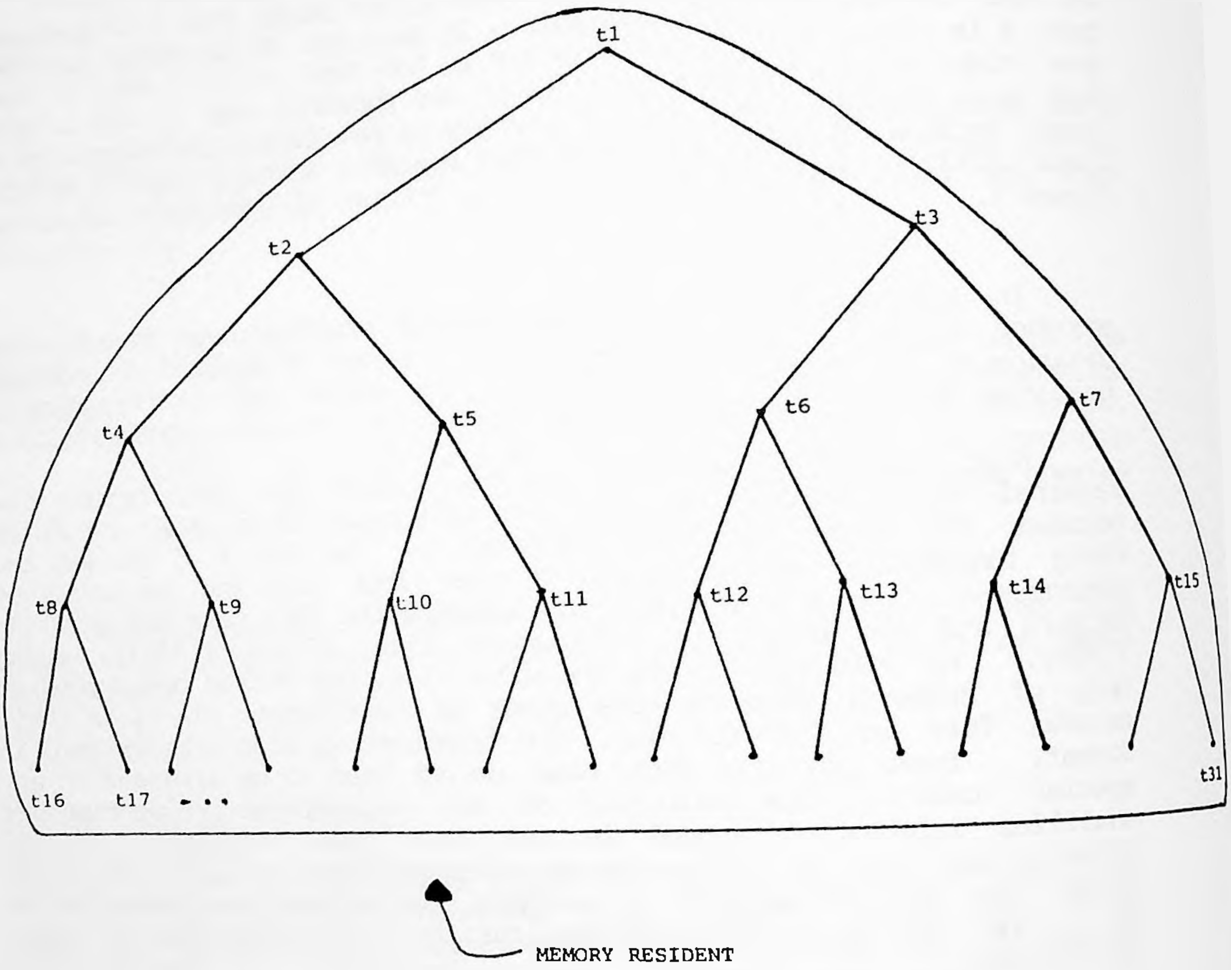


Figure 1.

Pro forma of an in-core PATRICIA-I P-tree. ti represents the twin bits. A branch path from ti to tj represents a chain of keys.

available, Morrison (1968). At the time R/5 was of the order of a few thousand. The algorithm was extended and enhanced to form a two-level and overlaid P-tree index, which significantly reduced this storage limitation, Clark (1972). PATRICIA-II was implemented on several large computers, e.g. CDC 6600 and 7600 and used extensively on very large data bases at the Los Alamos National Laboratory. PATRICIA-II was designed to build a lower index, which is always core resident, and an upper index, divided into pages which are stored on disk. Each upper index page references only itself and the data base. Both the lower and the upper index are P-trees. Second level P-trees are swapped in and out of memory as required; hence at most one page of the upper index, along with the lower index is core resident at any given time. Performing a transaction, i.e. an insertion, deletion or search, normally requires one disk access to fetch required pages of the upper level index into memory.

P-TREES OF TYPE PATRICIA-III

PATRICIA-III was implemented on the DEC System 10 and UNIVAC 1100 series on-line systems and is still in use in a parts data base environment, Thompson (1975). This implementation essentially uses the extended two-level P-tree algorithms developed for PATRICIA-II; however, the system was designed and implemented for an interactive on-line environment.

Recently the two-level P-tree structure has been extended to n-levels, thus removing the constraint of two-level indexing. In PATRICIA-II and III, at some point in time, a given second level page index could become full and a new second level page must be spawned. In essence this means the full page must be split, forming a dead part and a live part. The spawned page being the complement of its parent, i.e. dead nodes in the spawned page are live nodes in the parent page and vice versa. When the second level page spawning occurred, a pointer to the new page had to be added to the lower index, ultimately increasing the size of the core resident lower page. This approach yields P-tree structures which are "low in height", which may induce many second level pages, Clark (1972), Korbin (1983). PATRICIA-II and III are constrained by having the lower page memory resident as well as nonsplitable. For a pro forma of P-trees of type PATRICIA-II and III, see Figure 2.

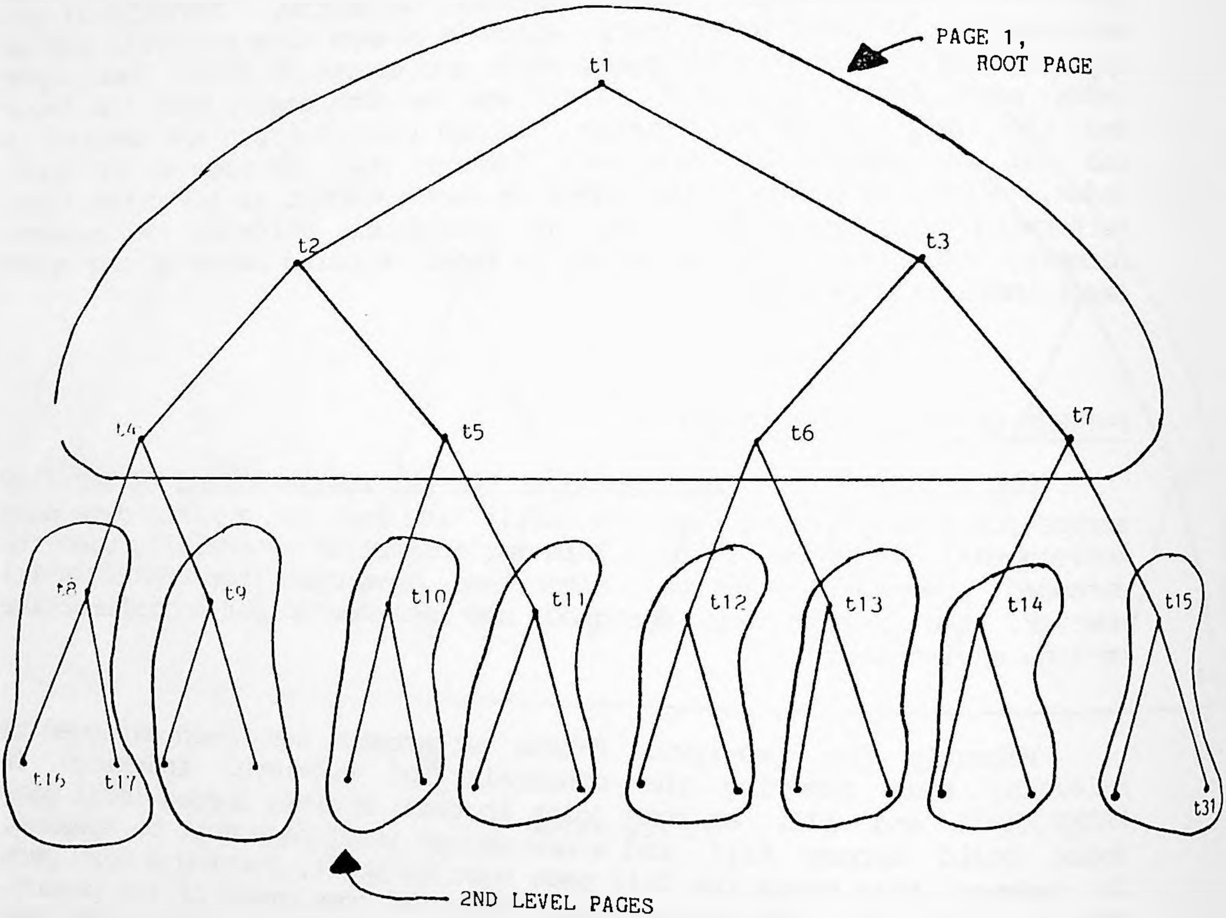


Figure 2.

Pro forma P-tree representation of P-trees of type PATRICIA-II and PATRICIA-III.

P-TREES OF TYPE PATRICIA-IV

In PATRICIA-IV, the algorithm has been enhanced to enable the P-tree to grow in a more normal manner. That is the lower P-tree, or root page, can also be split and spawn a new level of pages, along with a new root page. The original algorithm has now been fully extended, and implemented in PATRICIA-IV while at the same time retaining the inherent efficiency of the original algorithm. The method can now be applied without any of the original constraints on the number of keys, or the host machine resources.

Using multilevel indexing, each transaction normally results in one or two mass storage accesses(s) to fetch the final page for referencing the data base. In practice, this number increases very slowly with the size of the data base. In fact, increasing the number of keys by three orders of magnitude increases the average number of upper level page faults by one, Clark (1985, unpublished). As in the PATRICIA-I algorithm, the number of disk accesses is bounded above by the number of key occurrences in the data base for search and delete transactions. For insertion, the number of disk accesses is bounded by the number of key occurrences in the data being added to the data base, Clark (1972), Korbin (1983).

P-trees, or trees of type PATRICIA, exist in several similar data structures. To extend the two level P-tree, used in the PATRICIA-II algorithm, requires that the index make use of a multiple purpose data structure, Morrison (1983). This multiple purpose data structure is called a MAP, an acronym for matching and permutation. A MAP is a finite set in which each member, x, has a spouse and a successor. The spouse function is a matching and the successor function is a permutation. The only tabulation required to describe a map is the successor function. The only primitive structure change after initialisation is a successor swap. MAPS are efficient and accomodating hosts to several data structures including binary trees.

In a typical application, the use of a MAP host reduces the memory requirement for describing a data structure by a third to half, and the time for updating and transmission by a similar factor. This is a significant reduction; especially when viewed in a small workstation environment. The MAP specifically used in extending to n-level paged P-trees is similar to a pure binary tree. The only differences are that as well as each node having zero or two children, each tree node also has a spouse. Nodes are divided into two sets, internal nodes called women and external nodes called men. An internal node´s right child is defined by a function called successor, that is a permutation, Korbin (1983), Morrison (1983). The left child is defined to be the successor of her husband. The spouse relationship is constant, i.e. a marriage or matching. In essence the MAP need only record the successor of each node. This extended P-tree data structure enables the complete manipulation of the tree by successively exchanging the successors of two nodes.

The importance of the MAP data structure is realised when one considers the operations of adding or deleting material from the data base. New internal as well as external tree nodes must be created or released during the transactions of adding and deleting. In addition, mass storage released by the deletion of data base material must also be freed for potential reuse at a later date. These two types of transactions require that the algorithm be able to create as well as free for later use P-tree nodes and physical mass storage locations.

In practice, the only portion of the n-level P-tree which is memory resident is the P-tree root, (also called the "trunk"), and at most one other page, (called a limb), of the upper index. The trunk, (lowest level page), and every limb, (a higher level page), has a positive integer associated with it called the limb number. Limbs are swapped into memory when required. Limbs reference themselves or higher level limbs or the data base, Korbin (1983). For a pro forma of P-trees of type PATRICIA-IV, see Figure 3.

SUMMARY AND FUTURE DIRECTIONS

Empirical results show that the P-tree algorithm for searching is near optimal and performance is consistent with theoretical expectations, Clark (1972), Knuth (1973), Lagana (1980), Korbin (1985, to be published), Clark (1985, to be published). For PATRICIA-I through PATRICIA-IV, it must be noted that the method applies to any set of keys in any data base. Retrieval computational effort is independent of the order in which material is added to the data base. The binary search, e.g. applies only to an ordered list.

In general, trie traversal is quite efficient. The operations of inserting, deleting and establishing membership can be performed in time proportional to the length of the word involved, Aho et al (1983).

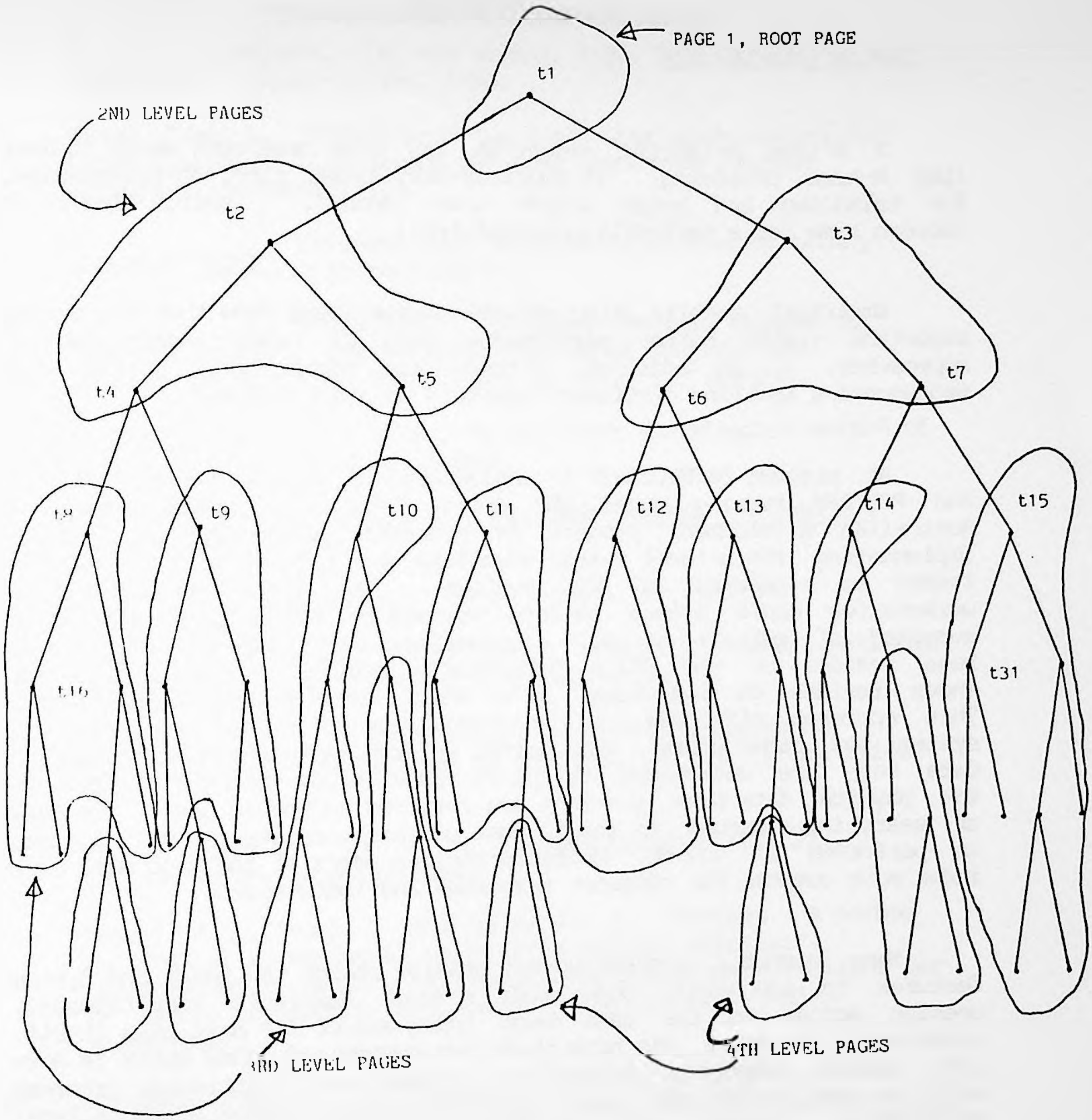
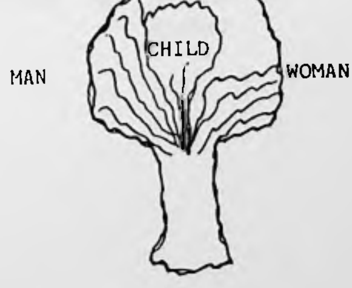


Figure 3.

Pro forma of P-trees of type PATRICIA-IV.



AN ENVIRONMENT FOR RETRIEVING TEXT

A P-tree balancing algorithm has been developed which is much like B-tree balancing. It requires only linear time, on the average, for balancing and keeps search time bounded. (search time)<1.5* (search time for a perfectly balanced tree).

Empirical results using variable length keys show that the P-tree algorithm yields better performance than B+ trees and/or hashing approaches. In addition, P-trees are particularly useful when implementing spelling dictionary/checkers on small systems.

At present PATRICIA-IV is implemented on the DEC VAX in PASCAL, C and FORTRAN running under VMS and/or UNIX. A joint Singapore and Australian development project is presently being carried out by implementing the n-level P-tree algorithm on a network of IBM type PC´s linked to a network of VAX systems. The system provides small workstation users access to large volumes of unstructured information composed of textual as well as numerical data. The distributed data base resides on the VAX´s mass storage devices. The n-level P-tree index resides on individual PC´s where users perform transactions. This approach off-loads the data base transactions and frees the main system for other users. When actual references to the distributed VAX data base are determined on the PC workstation, then the contents of the physical data base locations are returned to the PC user. The task of searching through an index, most of which resides on mass storage, is performed in the PC. Multiple on-line users of the same data base index must compete for computer resources and index access.

Many computer systems do not provide enough raw power and system features to adequately cope with several terminals simultaneously seeking access to the data base index and/or the data base itself. Individual PC users can have their own customised index which is down line loaded, providing access to information of particular interest held on the central data base. This allows distributed PC users local information storage and retrieval facilities far beyond the capabilities of personal computers. A similar approach could be used in a situation involving a network of small micro or personal computers.

Results show that this method is a very practical and economical means of providing a form of distributed data base capabilities to users.

REFERENCES

1. Aho, A.V., Hopcroft, J.E. and Ullman, J.D., Data Structures and Algorithms, Addison-Wesley, 1983.

2. Clark, James L., "PATRICIA-II Two level Overlayed Indexes for Large Libraries", Los Alamos National Laboratory, Report LA-5020-t, September 1972.

3. ______________. "Computational Performance of n-level binary P-trees", 1985 (to be published).

4. ______________. "PATRICIA-II. Two Level Overlaid Indexes for Large Libraries", International Journal of Computer and Information Sciences, Vol. 2, No. 4, 1973, PP: 269-292.

5. Cooke, Jeffrey L., "P-trees, An Efficient and Flexible Method of Sorting", 1985 (to be published).

6. Date, C.J., An Introduction to Database Systems - Volume II, Addison-Wesley, 1983.

7. Fredkin, E., "Trie Memory", CACM 3(1960), PP: 490-500.

8. Korbin, Christine M., "PATRICIA-User Reference", Sandia National Laboratories, Report SAND 83-0199, March 1983.

9. ______________. and Rowe, J.C., "PAT-IV Performance Statistics", to be published as a Sandia National Laboratories Report.

10. Knuth, Donald E., The Art of Computer Programming, Vol. 3, Addison-Wesley, Reading, Mass., 1973.

11. Lagana, M.R., Leoni, G. and Sprugnoli, R., "P-Trees - A method for digital accessing of paginated secondary memories", (Proceedings of Annual Conference AICA-Associazione Italiana per Il Calcolo Automatico ´80) October 1980, Bologna, Italy, PP:980-988.

12. Morrison, Donald R., "Practical Algorithm to Retrieve Information Coded in Alphanumeric", Sandia National Laboratories, Report SC-RR-67-734, October 1967.

13. ______________. "PATRICIA-Practical Algorithm to Retrieve Information Coded in Alphanumeric", J. ACM 15(4):514-534 (1968).

14. ______________. "Binatural numbers", J. Pi Mu Epsilson 5(6):276-278 (1972).

15. ________________. "MAPS-Multiple Purpose Data Structures", Tech Report No. CS-8(1983), Dept of Computer Science, University of New Mexico.

16. Pollard, J.P., Cawley, R.J., and Cergovska, J., "Text in Relational Database Applications", Proc. Australian Computer Conference, November 1984, Sydney, Australia.

17. Thompson, R.E., "PATRICIA-III Systems Description", Sandia National Laboratories, Report SAND 75-0242, April 1975.