LA-5020-T

THESIS

PATRICIA II. Two Level Overlaid
Indexes for Large Libraries

los alamos
scientific laboratory
of the University of California
LOS ALAMOS, NEW MEXICO 87544

UNITED STATES
ATOMIC ENERGY COMMISSION
CONTRACT W-7405-ENG. 36

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

This report is a printed version of a thesis submitted as partial fulfillment of the requirement for an advanced academic degree and as such represents the independent work of the author. It has not been edited by the Technical Information staff.

Printed in the United States of America. Available from
National Technical Information Service
U. S. Department of Commerce
5285 Port Royal Road
Springfield, Virginia 22151
Price: Printed Copy \$3.00; Microfiche \$0.95

LA-5020-T

Thesis

UC-32

ISSUED: September 1972

PATRICIA II. Two Level Overlaid Indexes for Large Libraries

by

James L. Clark

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

This report is derived from a dissertation submitted to the Department of Mathematics of The University of New Mexico in partial fulfillment of the requirements for the Degree of Doctor of Philosophy.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
1. INTRODUCTION.....	1
2. MOTIVATING CONCEPTS.....	4
2.1 Radix Sorting.....	4
2.2 Binatural Numbers.....	7
3. DEFINITION AND EXPLANATION OF THE BASIC VOCABULARY OF PATRICIA.....	11
4. ASSIGNING FURTHER MEANING AND NUMBERS TO STARTS, ENDS, BRANCHES, CHAINS, AND TWINS.....	17
5. TWO BASIC ALGORITHMS WHICH ARE ESSENTIAL TO PATRICIA'S PERFORMANCE.....	22
5.1 The FINDONE Algorithm.....	22
5.1.1 Library Metrization.....	23
5.2 The FINDALL Algorithm.....	30
6. DETECTING THE PRESENCE OF A PHRASE AND FINDING ITS PROPER OCCURRENCES WITH PATRICIA.....	32
7. BUILDING THE LIBRARY INDEX FOR PATRICIA.....	34
7.1 Permuting the Order of Adding a Collection of Books to a Library L.....	38
8. METHODS OF SATISFYING RULE 1.....	44
9. EXTENDING PATRICIA TO PATRICIA-II.....	45
10. DEFINITION AND EXPLANATION OF THE BASIC VOCABULARY OF PATRICIA-II.....	50
11. DETECTING THE PRESENCE OF A PHRASE AND FINDING ITS PROPER OCCURRENCES WITH PATRICIA-II.....	52
11.1 The LINK Algorithm.....	52
11.2 The FINDONE Algorithm.....	52
11.3 The FINDALL Algorithm.....	54

TABLE OF CONTENTS, continued

<u>Section</u>	<u>Page</u>
12. BUILDING THE LIBRARY INDEX FOR PATRICIA-II.....	56
12.1 The Lower Index.....	56
12.2 The Upper Index.....	57
13. THE ADD ALGORITHM FOR PATRICIA-II.....	58
13.1 The Upper Page SPLIT Algorithm.....	58
13.2 The Upper Page Dead Subtree PRUN Algorithm.	62
14. MULTIPLE KEYS AND USEFUL RETRIEVAL TECHNIQUES....	67
15. DISCUSSION OF USEFUL PROGRAMMING TECHNIQUES AND COMPUTER IMPLEMENTATION.....	68
16. APPLICATIONS.....	76
APPENDIX - Sample Input/Output.....	77
REFERENCES....	89

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1. Representations of a Right Library.....	15
2. A trivial binary tree.....	19
3. A non-trivial binary tree.....	19
4. Flowchart for FINDONE.....	24
5. Flowchart for FINDALL.....	31
6. Flowchart for BUILD INDEX algorithm.....	35
7. Flowchart for ADD algorithm.....	39
8. A trivial library and corresponding index.....	42
9. The binary tree representation of an in-core index generated by PATRICIA.....	48
10. A paged index generated by PATRICIA-II.....	49
11. Flowchart for LINK.....	53
12. Flowchart for PATRICIA-II ADD.....	59
13. A full upper page, P_5 , of index.....	60
14. Graph of upper page size versus real time in seconds to query entire library.....	71
15. Graph of upper page size versus real time in seconds to index entire library.....	72
16. Graph of real time to query entire library versus number of upper pages.....	73
17. Comparison of PATRICIA and the binary search..	75

ABSTRACT

PATRICIA is a Practical Algorithm To Retrieve Information Coded In Alphanumeric. The library index built by PATRICIA is entirely core resident. PATRICIA-II extends PATRICIA by building both a lower index and an upper index. This allows for the indexing of a much larger library. The lower index is core resident and the upper index is divided into upper pages which are stored on disk. At any given time only the lower index and one upper page are in core. Upper pages reference only themselves, not each other. In addition to these innovations, PATRICIA-II retains all of the features of its predecessor. When new material is added to the library, it is not necessary for the library text or the corresponding index to be rearranged.

When keys are presented to the system, the time spent in determining the presence or absence of a corresponding target is a function of the length of the key and the number of occurrences of the key. PATRICIA-II has been implemented as a FORTRAN program on the CDC-6600 and 7600.

PATRICIA-II. TWO-LEVEL OVERLAYED
INDEXES FOR LARGE LIBRARIES

1. INTRODUCTION

This paper describes the structure, organization, storage, searching, and retrieval of information. The term information connotes basic elements which can be processed or produced by a binary digital computer.

Libraries are constructed by librarians in a manner which allows the user to find, that is retrieve, the information he requires, in a regimented way. Examples include dictionaries, telephone directories, street addresses, zip code directories, and general libraries. The methods of indexing a library vary with the nature of the contents of the library.

In addition to this is the problem of a wide divergency of file layouts and types of retrieval requests. These types of problems seem to preclude the development of a truly general information retrieval language along the lines of ALGOL, FORTRAN, and COBOL [1,10].

Fortunately another solution presents itself. One can say that the user approaches the library with his own natural language. When we use the telephone directory we usually approach it armed with a name and street address. The natural language used in this case is the sequence (name and street address). The information to be retrieved from the directory

may be a phone number or street address, depending on the needs of the user.

Two basic concepts in the field of information retrieval (which will be denoted by IR from now on) are key and target [9]. The key is something the user has in his possession when he approaches the library in search of some type of information called the target. The target is something that he does not have completely in his possession when the library is approached with a key. It is the information that he is searching for, of which the key is a small portion.

In the phone book example above, the key is usually the sequence (name and street address) and the target is the phone number which corresponds to the unique key. In a given quest for information the key and target may be subjected to one or more iterations before the final desired information is found. That is, they may vary dynamically, so that new keys are generated and new or modified targets are sought. The librarian's job is to construct a directory, index, catalog, or whatever, to allow the user to submit keys and find targets corresponding to his keys without having to search through the entire library. The librarian should construct his index in a manner which allows the presentation and acceptance of keys in a way which is as natural as possible to the potential user. The library index should be parsimonious in the areas of space and ease of modification.

The amount of energy expended, regardless of the form, to query, add, or delete a target in a given library should remain bounded [4].

PATRICIA [4,5], developed by Donald R. Morrison, is a system for indexing, modifying, querying, and retrieving information in a binary coded library. PATRICIA is an acronym for "Practical Algorithm To Retrieve Information Coded In Alphanumeric." The usefulness of PATRICIA is limited because the index for a given library must be entirely contained in the host machine's main memory. This severely restricts the size of the libraries which are candidates for automation.

PATRICIA-II is an extension of PATRICIA which greatly reduces this limitation but does not completely remove it. The restriction is not completely removed because there are many problems which cannot be handled by existing machines. Library automation seems to be one of these problems. This extension consists of building the index in two parts, called the upper and lower index. The lower index is core resident and the upper index is stored on the disk with the text. The number of keys on which PATRICIA has been used is typically around 4000. PATRICIA-II has been run using 15000 keys.

2. MOTIVATING CONCEPTS

Before plunging directly into the explanation of PATRICIA, it will be helpful to examine two concepts which are closely related to the PATRICIA algorithm. The first concept is from the field of sorting and the second from an extension of the natural numbers.

2.1 Radix Sorting

A sorting algorithm is called a radix method if it is one which takes into account the representation of the items to be sorted. In the method of radix exchange, the unordered set of items is examined starting with the first entry, and only the most significant bit is examined [2]. If, during a forward scan of the list, the binary symbol¹ A is found in the most significant position of an element, a scan of the unordered items starts backwards from the last item. If the binary symbol B is found before reaching the previously found A the two entries are exchanged and the scan continues. Two pointers must be used during the scan in order to know when forward and backward scans have met. When the forward and backward scans meet, the unordered list of items has been split into two unordered lists of items, those whose most significant bit is B and those whose most significant bit is A. By the process of recursion, the entire unordered list will eventually be sorted.

¹A and B will be used for 0 and 1 respectively to avoid confusion later on.

The idea behind PATRICIA is similar. It is a radix method of retrieving information in which the radix is 2. When a user's key is encoded in a binary computer it will begin with an A or B. By merely examining the first bit of a presented key, PATRICIA can determine which part of the index to begin examining.

One might ask why the alphabet on which PATRICIA works is restricted to binary. To answer this it may help to look again at the radix exchange method of sorting. The primary reasons for restricting the radix exchange method to operate on items represented in the base 2 are simplicity and efficiency. For example, if a list of alphabetic information were to be sorted, the items in the list could be represented in radix 26. The list could then be ordered by distributing the items into 26 piles, where the contents of a given pile are all items which begin with the same letter. The process is then applied to the contents of each of the 26 piles, where this time the second letter is examined, and so on.

The question of selecting a proper alphabet on which PATRICIA should operate has essentially the same answer. The allocation of storage and efficiency of inner loops are made easier and more efficient if the alphabet is binary [5]. The index constructed by PATRICIA is of a very special type and is designed for a special kind of user, namely a binary computer. The index contains only numbers which indicate locations in the text or index [5, p. 515]. When keys are presented to the

system, the time spent in determining the presence or absence of a corresponding target is a function only of the length of the key and the number of occurrences of the key.

Fortunately the library size does not enter into the computation time. It is moderately easy to add new information to the text of the library and make the necessary modifications in the existing index to show the presence of the new information. This process consists of: first, adding the new material to the existing text; then determining how much, if any, of the new material is indicated as already indexed; and finally, adding five numbers and changing another number in the previous index.

When new items are added to the text of an existing library, it is not necessary to relocate any of the old text. In fact, the only restraints imposed on the library text are those which were already prescribed. As was mentioned above, the index constructed by PATRICIA consists only of numbers. Consequently the keys appear where they should, directly in the library text as subsets of targets. Keys occur in their natural form, and their sizes are limited only by the storage space which limits the text size. The librarian can designate that any given target be retrievable by as many keys as he may select in it. He can also designate that a single key retrieve many targets.

The above features of PATRICIA remove the burden of querying an automated library in a way which is unnatural for the user.

2.2 Binatural Numbers

In order to gain more insight into the PATRICIA algorithm an extension of the natural number system will be discussed. Of all the mathematical number systems the natural numbers, denoted by N , are the most fundamental. The system N is characterized by the following three axioms in terms of a first element 1 and a successor function S , under which N is closed.

N1. S is one-to-one.

N2. 1 is not in the range of S .

N3. The only subset of N which includes 1 and is closed under S is N .

The function S is the counting function $S(n) = n+1$. A very useful and important property of N is the existence of a predecessor function $P(n)$. The function $P(n)$ is defined on all of the natural numbers except 1 by:

$$P(S(n)) = n.$$

The system N can be extended to a system $\bar{B}$, called the binatural numbers, which also has a first element 1 [7]. The successor function S of the system N has, as its counterpart in $\bar{B}$, a pair of successor functions, denoted by L and R , called the left successor and the right successor. The system $\bar{B}$ is closed under each of these functions. Furthermore, the system $\bar{B}$ satisfies the following three axioms:

- $\bar{B}1$. L and R are both one-to-one.
- $\bar{B}2$. The ranges of L and R are disjoint, and neither includes 1.
- $\bar{B}3$. The only subset of $\bar{B}$ which includes 1 and is closed under both L and R is $\bar{B}$ itself.

As in the system N , $\bar{B}$ also has a predecessor function P , which is defined for every binatural number except 1 by:

$$P(L(n)) = P(R(n)) = n.$$

The system $\bar{B}$ has many representations. These representations differ not only in form but also in use. Now consider the following two different definitions of L and R and the two different forms of $\bar{B}$ to which they give rise.

Example 1

$\bar{B}$ is the set of natural numbers. 1 is the first number. $L(n) = 2n$. $R(n) = 2n+1$. The range of L is the set of even natural numbers. The range of R is the set of odd natural numbers > 1 . The predecessor function, $P(n)$, is defined as:

$$P(n) = \lfloor n/2 \rfloor$$

where $\lfloor n/2 \rfloor$ denotes the integral part of $n/2$. Schematically the system $\bar{B}$ is as follows:

Example 2

$\bar{B}$ is the set of all finite sequences, including the empty or null sequence, Ω , of A's and B's. The first sequence is Ω . $L(S) = SA$, the concatenation of the string S , containing some combination of A's and B's, and the symbol A. $R(S) = SB$, the concatenation of S and B. The range of L is the set of strings whose last term is the symbol A. The range of R is the set of strings whose last term is the symbol B. The predecessor function, $P(S)$, is the truncation of the rightmost symbol of S . Schematically the system $\bar{B}$ is as follows:

The above example is the free monoid with two generators, that is on the set $\{A, B\}$, which plays a role in the coding of information for binary digital computers. The LEVEL of a binatural number n is defined to be the number of steps which must be taken up the binary tree to reach n . It is defined inductively by:

$$\text{LEVEL}(1) = 1$$

$$\text{LEVEL}(L(n)) = \text{LEVEL}(n) + 1$$

$$\text{LEVEL}(R(n)) = \text{LEVEL}(n) + 1$$

In the first example

$$\text{LEVEL}(n) = 1 + \lceil \log_2(n) \rceil$$

and in the second example

$$\text{LEVEL}(S) = \text{length } (S).$$

3. DEFINITION AND EXPLANATION OF THE BASIC VOCABULARY OF PATRICIA

The vocabulary of PATRICIA, along with the necessary terminology and symbolism, will now be introduced [4,5,6].

Definition 1

A right library is a library which is read from left to right and includes with every phrase all its left parts.

The text of a right library consists of a finite sequence of bits, each represented by the binary symbol B or A. Each position in the text is identified by a positive number, which is called the bit address or simply the address. Each position in the text is occupied by an A or a B. In the set of text addresses there are two important subsets. These addresses are not marked in the text itself, but they are a part of the index. This leads us to the following two definitions.

Definition 2

A start designates a place in the text which is the beginning of a potential target.

Starts are chosen by the librarian in a way which, hopefully, will lead to the finding of targets. Symbolically starts will be represented by S_i , for some specified i .

Definition 3

A stop is a text address which indicates the rightmost address of a potential target.

A consecutive sequence of text in a right library will be called an occurrence. PATRICIA detects a particular kind of occurrence, called a proper occurrence.

Definition 4

A proper occurrence is an occurrence which begins at a start and ends not later than the next stop.

Definition 5

An L-phrase is a consecutive sequence of text which has at least one proper occurrence.

Definition 6

The library is defined to be the set of all L-phrases and will be denoted by L . The shortest L-phrase in any library L is the empty or null phrase and is denoted by Ω . It follows that PATRICIA finds only proper occurrences due to the way in which the library was defined (Definition 6).

Definition 7

An end, in L , is defined to be an L-phrase which begins at a start and terminates at the next stop.

In the sequel, ends will be denoted by E_i , for some i . As will be seen in the next section, i is always an odd integer.

Definition 8

A book is defined to be an end which is not a right part of another end.

An alternative way of stating the definition of a book is as follows:

An end is a book if the start at which the end begins is leftmost among the starts which correspond to its stop.

For a given L-phrase p , in a given L , one can form two minimal right extensions of p , namely pA and pB . The phrase p is an end in L if neither pA nor pB is in L . When both minimal extensions, pA and pB , are in L , we are led to the following definition.

Definition 9

If both minimal right extensions, of a given L-phrase $p \in L$, are in L , then p is called a branch in L .

An alternative way of saying this is that a branch occurs when pA and pB are left parts of two different ends.

Definition 10

When a branch occurs in L , the two resulting phrases, pA and pB , are called twin L-phrases in L .

There is one special case which occurs here: Ω is also classed as a twin. The reason for this will become apparent in the next section.

Definition 11

The set of L-phrases whose members have proper occurrences which begin at the same starts are called chains.

In the sequel, branches, twins, and chains will be denoted by $\tilde{B}_i$, T_j , and C_k , respectively, for some i , j , and k .

This completes the definition of PATRICIA's vocabulary.

In order to illustrate the above definitions, the reader should now refer to the example in Figure 1 (next page). In the figure, starts are denoted by $($, stops are designated by $)$, and L-phrases are denoted by an $\uparrow$ beneath them. The set of proper occurrences appearing in the text of the example is the union of the following sets:

$$S_1 = \{ABB, ABBA, ABBAB, ABBABA, ABBABAB, ABBABABB\}$$

$$S_2 = \{A, AB\}$$

$$S_3 = \{ABA, ABAB, ABABB\}$$

$$S_4 = \{\Omega\}$$

$$S_5 = \{BABA, BABAB, BABABB, BABABBA\}$$

$$S_6 = \{B, BA, BAB\}$$

$$S_7 = \{BABB, BABBA\}$$

The ends occurring in the example are the last elements of sets S_1 , S_3 , S_5 , and S_7 . The only books in L are the last elements of S_1 and S_5 . The reason for this can be seen by examining the text table in Figure 1. One can see the four ends ABBABABB, ABABB, BABABBA, and BABBA. By referring back to the definition of a book, ABABB and BABBA are not books because the starts at which they begin are not leftmost among the starts corresponding to their stops. The branches in L are Ω , AB, and BAB. The resulting twins are A, ABB,

Figure 1.
Representations of a Right Library.

ABA, B, BABA, BABB, and, as was mentioned above, Ω . The chains which are in L are just the sets described above, $S_1, S_2, S_3, S_4, S_5, S_6$, and S_7 . Notice that the first elements of these seven sets are the twins in L.

The alert reader will have noticed a serious flaw with the development that has been presented so far. If one asks the question about the uniqueness of the ends which appear in L, one wonders how PATRICIA would distinguish between non-unique occurrences of ends. The answer is that it cannot make this distinction. Thus an inherent rule emerges and provides the obvious constraint. Stated briefly with further explanation and clarification deferred until Section 8, it is as follows [4,5]:

Rule 1

Every end appearing in a given library L is restricted to having a unique proper occurrence in the library text. This unique proper occurrence begins at one and only one start and terminates at the next stop.

4. ASSIGNING FURTHER MEANING AND NUMBERS TO STARTS, ENDS, BRANCHES, CHAINS, AND TWINS

In the interest of heuristic motivation a study of Figure 1 shows that it resembles a two-dimensional graph where the x-axis, or axis of abscissas, denotes the chain numbers and the y-axis, or axis of ordinates, denotes the length of the L-phrases in L. It follows that every L-phrase in L can be uniquely identified by the ordered pair (chain number, length of the L-phrase p). The length is the number of binary symbols necessary to represent p in a binary machine and is denoted by $\ell(p)$.

It is more difficult to determine the chain number, or more precisely, the chain number for a given p, which will be denoted by $c(p)$ and referred to as the chain coordinate of all L-phrases in chain c. Returning to the definition of a chain, we see that the concept of starts plays a role. To be more specific, a chain is a set of L-phrases that has proper occurrences beginning at the same set of starts. By the definition of ends, it is clear that the longest proper occurrences in some of the chains must be ends. Similarly, by reviewing the definition of branches, the remaining chains must have branches as their longest members. It is also clear that the shortest member of a chain must be a twin. If we could assign one set of unique numbers to all ends and another set of unique numbers to all branches, then unique numbers would be assigned to all chains. In order to assign

numbers to ends, it is necessary to appeal to the definition of an end and observe that starts and ends are in 1-1 correspondence. By Rule 1, each end has a unique proper occurrence in the text. Hence the problem of assigning numbers to ends has been reduced to a problem of assigning unique numbers to starts. Recall that the set of chains that have ends as longest members and the set of chains that have branches as longest members are disjoint. If we are going to assign numbers, our intuition might say to assign even and odd numbers, therefore, let us denote by α the set of chains that have ends as longest members and by β the set of chains that have branches as longest members. If α is enumerated by odd positive integers, then all of the chains in α will be assigned a unique odd positive integer. At this time let us say that starts will be assigned unique positive odd integers. What is needed now is a vehicle for assigning unique positive even integers to branches and hence to chains of which they are longest members. In order to do this, the following definition is needed. A stem of an end E_n , for some odd $n > 1$, is the longest left part of E_n , which is also a left part of E_m for some odd $m < n$. At this point it may help to digress for a moment and consider the simple binary tree in Figure 2.

Figure 2.
A trivial binary tree.

E_1 and E_3 represent terminal nodes and B_2 represents the root of the tree. If E_1 and E_3 are viewed as ends and the definition of a stem is applied, the stem of E_3 would be B_2 . If the binary tree in Figure 3 is considered

Figure 3.
A non-trivial binary tree.

and the same analysis is applied, we would obtain:

The stem of $E'_3 = B'_2$

The stem of $E'_5 = B'_4$

The stem of $E'_7 = B'_6$

If we define branch numbers as

B_{n-1} = the stem of E_n

then, since n is an odd positive integer, consecutive even positive integers are assigned to branches. Since we have enumerated α with unique positive odd integers and β with unique positive even integers, every chain in L has been assigned a unique positive integer. It follows that every L -phrase in L has been assigned a unique ordered pair $(c(p), l(p))$, which will serve as the coordinates for locating proper occurrences of the phrase p .

So far, numbers have been assigned to starts, ends, branches, and chains. What we need now is a way of assigning numbers to twins. Since twins are related to branches in a prescribed manner and positive even numbers have been assigned to branches, the natural thing to do is to assign numbers to twins as follows:

$$T_n = \tilde{B}_n A$$

$$T_{n+1} = \tilde{B}_n B$$

In this way, since 2 is the smallest branch number, twins are assigned positive integers greater than one. This is why we defined $T_1 = \Omega$, as was pointed out in the previous section. It is clear that the set of numbers assigned to chains is the same set of numbers assigned to twins. Although the twin and chain numbers range over the same set, it is not true that the number assigned to a twin is the same as the number assigned to its chain. For this reason a table is needed in the index which, for a given twin number,

indicates the corresponding chain number. This table is called the TWIN INDEX or the twin-chain coordinate transformation TC.

The next entry in the index of a given L is the HEIGHT INDEX. In this table the length of the longest members of the chains is tabulated. At this point it is germane to make two observations. First, since positive odd integers were assigned to starts, and consequently to ends, if an L -phrase p has an odd chain number n assigned to it, then there exists a unique start, S_n , at which a proper occurrence of p begins. Second, if an L -phrase p has an even chain number assigned to it, then there are at least two proper occurrences of p in L , and one of them is at S_{n+1} .

5. TWO BASIC ALGORITHMS WHICH ARE ESSENTIAL TO PATRICIA'S PERFORMANCE

These two algorithms are called FINDONE and FINDALL. They are the two most important algorithms in PATRICIA's repertoire of algorithms. The purpose of FINDONE is to output the chain coordinate of one proper occurrence of a phrase p . The purpose of FINDALL is to output the chain coordinates of all proper occurrences of a phrase p . Further analysis of these two algorithms is as follows in the next two sections.

5.1 The FINDONE Algorithm

The purpose of FINDONE is to output a chain coordinate c , which is the chain coordinate of a given phrase p , if p is in L . The algorithm does this by looking at selected bits of the phrase p and at the twin and height indexes of L . Each execution of the algorithm generates a sequence of pairs (t, c) which are the twin numbers and chain numbers of the twin left parts of p in increasing order of length. If p is in L , the last c generated by FINDONE is the chain number of the phrase p . The start S_c is a start at which one proper occurrence of p begins in the library text. If c is even, there is more than one proper occurrence of p ; if c is odd, there is only one proper occurrence of p . If there is no proper occurrence of p in L , then the longest phrase in chain c shares with p a common left part, which is the longest left part of p in L . That is, FINDONE finds the L -phrase in L which is closest to the phrase p . The meaning of the term closest is pursued

in much greater detail in Section 5.1.1. For a flowchart of the FINDONE algorithm refer to Figure 4.

5.1.1 Library Metrization

In this section we will introduce a metric function which enables us to determine the closeness of L-phrases. It will be shown that, in the sense of the metric, FINDONE finds the chain number of the closest phrases in L to the query phrase p.

Consider P, a set of finite phrases which defines a right library L. Now introduce another symbol, *, such that $* \neq A$ and $* \neq B$. Now construct another set of phrases by taking a phrase $p \in P$ and defining a new phrase p^* by

$$p^* = p***...$$

Thus, p^* is obtained by an infinite extension of p by the symbol *. If this is done for every $p \in P$, we obtain a new set of phrases of infinite length. This new set of phrases will be denoted by $\bar{P}$.

Every phrase $p^* \in \bar{P}$ may be viewed as an infinite string comprised of some combination of the symbols A, B, and *.

Examples include

$$p^* = BBB...B***...$$

$$p^* = ABAAAABA***...$$

$$p^* = AAA...A***...$$

Figure 4.
Flowchart for FINDONE

and of course the empty or null phrase, Ω , is extended to

$$\Omega^* = ***...$$

Each position in a phrase is identified by a number called its address. Phrase addresses are positive integers beginning with one and increasing as the phrase is read from left to right. Each position in a $\bar{p}^*$ phrase is occupied by either an A, B, or *. The following example will help to illustrate the above discussion.

Example 1

$$\begin{array}{l} \bar{p}^* = \text{ABBAAA***...} \\ \quad \quad \quad 123456789 \end{array}$$

The i th position in the phrase $\bar{p}^*$ of Example 1 contains one of the three symbols A, B, or *. For example, when $i = 7$ (i.e. position 7), we find a *.

The notation $\bar{p}_i^*$ will be read as "the contents of the i th position of the phrase $\bar{p}^*$."

It is well known that a nonempty set X can always be made into a metric space by what is known as the trivial metric [11]. That is, by defining the metric function d by:

$$d(x,y) = \begin{cases} 1 & x \neq y \\ 0 & x = y \end{cases} \quad x,y \in X.$$

We will now define a function D which maps $\bar{P}^* \times \bar{P}^* \rightarrow R^+$. Let the distance, D , between two arbitrary phrases $\bar{p}^*$ and $\bar{q}^*$ be

$$D(\overset{*}{p}, \overset{*}{q}) = \sum_{i=1}^{\infty} d(\overset{*}{p}_i, \overset{*}{q}_i) 2^{-i},$$

where $d(\overset{*}{p}_i, \overset{*}{q}_i)$ is the trivial metric on the set $\{A, B, *\}$.

In order to show that the function D is a metric it must be shown that [11]:

- (1) $D(\overset{*}{p}, \overset{*}{q}) \geq 0$ and $D(\overset{*}{p}, \overset{*}{q}) = 0$ iff $\overset{*}{p} = \overset{*}{q}$
- (2) $D(\overset{*}{p}, \overset{*}{q}) = D(\overset{*}{q}, \overset{*}{p})$
- (3) $D(\overset{*}{p}, \overset{*}{r}) \leq D(\overset{*}{p}, \overset{*}{q}) + D(\overset{*}{q}, \overset{*}{r}) \quad \forall \overset{*}{p}, \overset{*}{q}, \overset{*}{r} \in P$

Proof:

Since $d(\overset{*}{p}_i, \overset{*}{q}_i)$ is the trivial metric on the $\{A, B, *\}$ it is clear that D satisfies (1) and (2). Furthermore, since $\forall i$

$$d(\overset{*}{p}_i, \overset{*}{r}_i) \leq d(\overset{*}{p}_i, \overset{*}{q}_i) + d(\overset{*}{q}_i, \overset{*}{r}_i)$$

which implies that

$$\sum_{i=1}^{\infty} d(\overset{*}{p}_i, \overset{*}{r}_i) \leq \sum_{i=1}^{\infty} d(\overset{*}{p}_i, \overset{*}{q}_i) + \sum_{i=1}^{\infty} d(\overset{*}{q}_i, \overset{*}{r}_i)$$

that is, D must satisfy (3).

Q.E.D.

If E is a subset of a metric space (X, D) , then the diameter of E , $\delta(E)$, is defined as follows [11]:

$$\delta(\phi) = 0;$$

$$\delta(E) = \sup \{D(x, y) \mid x, y \in E\}$$

when $E \neq \emptyset$ and the supremum exists; and $\delta(E) = \infty$ otherwise.

It is interesting to note that a measure of the closeness of elements of an arbitrary chain, c , in a right library L is given by

$$\delta(c) = \sum_{i=l(s)+1}^{l(b)} 2^{-i} ;$$

where $l(s)$ is the length of the shortest member and $l(b)$ is the length of the longest member of chain c . It is also interesting to note that for an arbitrary finite right library L the least upper bound of $\delta(L)$ is

$$\sum_{i=1}^{l(b)} 2^{-i} ,$$

where $l(b)$ is the length of the longest phrase in P .

Theorem

The chain number returned by FINDONE, operating on $p \notin L$, is the chain number of the closest L -phrases to p . Furthermore, there may be many closest L -phrases but they are all in the same chain.

Proof:

Let $p \notin L$ be the query phrase. Suppose q and r are two L -phrases which satisfy

$$D(p,q) = D(p,r) ,$$

i.e., both are equally close to p and are such that no phrase in L is closer to p than q and r .

Without loss of generality suppose $\ell(q) \leq \ell(r)$. Then $\ell(p) \geq \ell(q)$, for suppose not. That is $\ell(p) < \ell(q)$. Let q' be the left part of $q \ni \ell(q') = \ell(p)$. Then

$$D(p, q) = \sum_{i \in K} 2^{-i}$$

and

$$D(p, q') = \sum_{i \in J} 2^{-i}$$

where K is the set of indices at which p differs from q and J is the set of indices at which p differs from q' . Since q' is a proper left part of q and since $\ell(q') = \ell(p) < \ell(q)$ it follows that

$$J \subset K \quad .$$

Therefore

$$D(p, q) - D(p, q') = \sum_{i \in K} 2^{-i} - \sum_{i \in J} 2^{-i} > 0 \quad ,$$

hence $D(p, q) > D(p, q')$. This contradicts the assumption that no phrase in L is closer to p than q and r . Therefore

$$\ell(q) \leq \ell(p) \quad .$$

Now suppose that $i \leq \ell(q)$. Then either

$$q_i = p_i = r_i$$

or both q_i and r_i are different from p_i . Since all three are either A or B, it follows that $q_i = r_i$. Therefore q is a left part of r .

We will now prove that q and r are in the same chain. If they were not, then let s be the longest member of q 's chain. s is a branch. One of the twin extensions t , of s , is a left part of r . The other one, t' , agrees with p at bit position $i = \ell(s) + 1$. To see this let $i = \ell(s) + 1$. Therefore $q_i = *$ and either

$$(1) \ p_i = * = r_i$$

or

$$(2) \ * \neq p_i \text{ and } r_i \neq p_i \ .$$

Since s is a branch, (1) is impossible. Therefore $p_i = A$ or B and $r_i = \bar{p}_i$, where $\bar{p}_i$ is the complement of p_i . Hence p agrees with t' at bit position i and t does not. Therefore

$$D(p, t') < D(p, r) \ .$$

This contradicts the assumption that no phrase in L is closer to p than q and r . Hence q and r are in the same chain.

We will now show that FINDONE returns the chain number of the closest L-phrases to p . FINDONE selects, during each execution of the inner loop, the twin, t , which agrees with p at bit position $i = \ell(t)$. The twin, t' , which disagrees with p at bit position $i = \ell(t) = \ell(t')$ is discarded. Because of the way in which the metric function, D , was defined, extensions of the rejected twins or their left parts, and extensions of the longest member of the chain, selected by FINDONE, must all be further from p than the phrase found by FINDONE. Consequently when FINDONE terminates, the closest twin L-phrase has been selected. Since twins are in one to one correspondence with chains, FINDONE returns the chain number of the closest L-phrases.

Q.E.D.

5.2 The FINDALL Algorithm

The algorithm FINDALL is called if the last chain number output by FINDONE is even. In this case there is more than one proper occurrence of the phrase p in L . FINDALL operates on this even chain number c and on the twin index of L to generate a sequence of odd chain coordinates, $c_1, c_2, \dots, c_k$, which are the chain coordinates of all ends which are right extensions of the phrase p , in alphabetical order. The beginnings of all proper occurrences in L of p are given by the sequence of starts, s_{c_i} . See Figure 5 for a flowchart of FINDALL.

Figure 5.
Flowchart for FINDALL.

6. DETECTING THE PRESENCE OF A PHRASE AND FINDING ITS PROPER OCCURRENCES WITH PATRICIA

For a given phrase p PATRICIA will examine the library index and the library text in the following manner [4,5].

- (1) FINDONE is executed. The input to the algorithm consists of the phrase p , the height index, and the twin index. The algorithm outputs a chain coordinate, c , which is the chain coordinate of the phrase p , if p is in L .
- (2) Examine the text at the start address S_c . If an occurrence of the phrase p is found which begins at this address, then p has a proper occurrence in L . If there is no occurrence, then p does not have a proper occurrence in L .
- (3) If the results of (2) indicate no proper occurrence of p in L , exit this sequence; otherwise, proceed to (4).
- (4) Test the parity of c . If c is odd we are finished, because there is only a possibility of one proper occurrence and this has already been found in (2). If c is even then we must execute the FINDALL algorithm. The input consists of the even chain coordinate c and the twin index. The output is a sequence of odd chain coordinates $c_1, c_2, c_3, \dots, c_n$. The phrase p has proper occurrences beginning at the text addresses, $S_{c_1}, S_{c_2}, \dots, S_{c_n}$.

The only place that PATRICIA examines the text for a key p , without knowing in advance if p is there, is in (2). If it is not found we have looked at the text only once and in that case only in one place.

When fast core storage is at a premium, and this is usually the case in any information retrieval problem, this is a major concern. For example, if the text is stored on some slower device like tape, disk, drum, or a photostore, one must minimize the number of times the text is examined. In preparing to examine the text, PATRICIA must look at the index. FINDONE must examine the height and twin indexes and FINDALL must have access to the twin index. For this reason, this portion of the index should be stored in fast core. The start index is not as critical as the other two indexes since it is not examined as often.

7. BUILDING THE LIBRARY INDEX FOR PATRICIA

BUILD INDEX is an algorithm for constructing the library index [4,5]. It begins executing with a trivial library and its corresponding index and is executed each time a book is added to the library. Before a book is added to the library, it is first examined by an algorithm called FINDSTART which is used to identify start symbols within a given book. The flowchart for BUILD INDEX is in Figure 6.

The algorithm ADD modifies the text and index of an existing library L to form a text and index for the extended library, say L'. It is executed once for each start symbol encountered by FINDSTART. The input to the algorithm consists of the phrase to be added, the next available text address, and the index for L. The output is the new library L'. The necessary steps are as follows:

- (1) Assume that an index for L exists and for some odd n a set of starts has been defined, say $S_1, S_3, S_5, \dots, S_n$. If the phrase to be added, say p , is a right part of a phrase already in the library text and has been designated by FINDSTART as a key whose occurrence should be recognized as a proper occurrence, then proceed to step (2). If the phrase p is not already in the library text, then it should be added to the existing text (at any place where there is room).

Figure 6.
Flowchart for BUILD INDEX algorithm.

- (2) The occurrence of the phrase p is made a proper occurrence by designating p as the new end, E_{n+2} . Along with this the starting address of p , in the text, is recorded in the start index at start S_{n+2} . Now proceed to step (3).
- (3) Execute the FINDONE algorithm with the phrase p as input. The output is a chain coordinate $c = c(p')$ where p' is the end, E_c , which has in common with p a longest left part, say q . This is the longest left part of the phrase p that is in L . To find the length of q , which is the number of bits of agreement of p and p' , p and p' are compared. Two dangerous possibilities arise here, both of which are violations of Rule 1.
- (i) If p is already in L the comparison of p and p' will show that p is a left part of p' , hence p already has a proper occurrence in L .
 - (ii) The phrase p' may be an end already in L and a left part of the phrase p . This is revealed when c is odd and $l(q) = \text{HEIGHT}(c)$. If a proper occurrence of p were added there would then be a second proper occurrence of p' . See Section 8 for avoiding (i) and (ii).

If neither (i) nor (ii) occurs, then p and p' are both extensions of q . One is an extension of qA and the other of qB , and only one of these is in L . Now proceed to step (4).

- (4) If $\ell(q)$ is less than $\text{HEIGHT}(c) = \ell(p')$ and $\ell(q)$ is less than $\ell(p)$, the height index is modified by entering

$$\text{HEIGHT}(n+1) \leftarrow \ell(q)$$

$$\text{HEIGHT}(n+2) \leftarrow \ell(p)$$

This step, along with step (2), has designated p and q , respectively, as the new end, E_{n+2} and its stem, branch $\tilde{B}_{n+1}$.

- (5) At this point it is necessary to execute FINDONE with q as input. The presence of the phrase q has already been established in step (3), hence the output of FINDONE is the ordered pair (t, c) where c is the chain coordinate of q in L and t is the twin coordinate of the shortest member of chain c . Adding p to L has made q a branch in L' and has split chain c into two chains in L' . The phrases in chain c of L which are longer than q are still in chain c of L' . The shortest member of that chain is the twin qx , where x is A or B as determined in step (3). The members of the chain c of L which are shorter than qx make up the new chain, say $c^* = n+1$, of L' , and q is the longest member of that chain.

This is recognized by replacing the c which occurs in the TWIN INDEX as $TC(t)$ by c^* .

- (6) Finally the twin index must be modified by entering c and $c^* = n+2$ as the new chain coordinates of the newly acquired twin extensions $qA = T_{n+1}$ and $qB = T_{n+2}$ of the new branch q . Which of the two new twin extensions receives which chain coordinate depends on which of the twin extensions, qA or qB , was already in L and which one has just appeared in L' as a left part of the phrase p .

The flowchart for the ADD algorithm is in Figure 7.

7.1 Permuting the Order of Adding a Collection of Books to a Library L

In this section a theorem will be presented on the shape of the binary tree, which is the data structure that describes PATRICIA's index. Before stating the theorem, it is necessary to introduce some terminology [3].

Definition 13

Two binary trees, T_1 and T_2 , are called similar if they are both empty or both nonempty, and if their left subtrees and their right subtrees are respectively similar.

Roughly speaking, similarity of binary trees means that they have the same shape.

Definition 14

Two binary trees, T_1 and T_2 are equivalent if they

Figure 7.
Flowchart for ADD algorithm.

are similar, and if, in addition, corresponding nodes contain the same information.

Now we are in a position to state the following theorem.

Theorem

Permuting the order of adding a collection of books to a library L changes the labeling of the binary tree, which describes the index, but produces similar binary trees.

Before proceeding to the proof of the theorem, the reader might find it useful to study the example in Figure 1 of Section 3 and make a correlation with the binatural numbers.

Proof:

If we take the definitions of T_n and T_{n+1} for the left successor and right successor functions of the free monoid, with generators A and B , it follows that we can construct a subtree of the infinite binatural tree for strings (see example 2 of Section 2.2). More specifically, if the example given in Figure 1 of Section 3 is used, we get the following tree-like structure:

If we expand the above structure to a binatural tree, the following binary tree is obtained.

Obviously the above tree is a subtree of the infinite binary tree for the binatural tree of strings. It is also clear that the above analysis can be performed on any binary tree which describes a PATRICIA index. Clearly, a unique path leads to each terminal node (i.e., each end) of the binary tree resulting from expanding a binary tree which describes a given PATRICIA index. Since the binary tree which describes a PATRICIA index is the union of these unique paths, and since the operation of union is independent of order, the theorem is proved.

Q.E.D.

A simple counterexample demonstrates that the work done by the ADD algorithm is not independent of the order of adding phrases to a library L . We define a unit of work as one cycle of the inner loop of the FINDONE algorithm. The text and index of a trivial library, L , are given in Figure 8.

TEXT			
1	2	3	4
5	6	7	8
A	A	A	B
B	B	B	B

START INDEX	
$S_1 = 1,$	$S_3 = 5$

TWIN INDEX	
Twin	1 2 3
Chain	2 1 3

HEIGHT INDEX	
Chain	1 2 3
Height	4 0 4

Figure 8.

A trivial library and corresponding index.

Assume that two phrases $p_1 = AAAB$ and $p_2 = AABA$ are to be added to L . If the order of addition is p_1 followed by p_2 the following units of work are done in the ADD algorithm.

- (1) The first execution of FINDONE, with p_1 as input, requires two units of work.
- (2) The subsequent execution of FINDONE, with $q = AAA$ as input, requires two units of work.

(3) The execution of FINDONE, with p_2 as input, requires three units of work.

(4) The subsequent execution of FINDONE, with $q = AA$ as input, requires two units of work.

Therefore the total work units done by ADD, using the order p_1 followed by p_2 , is nine.

If the order of addition is reversed, i.e., p_2 followed by p_1 , the following results are obtained.

(1) The first execution of FINDONE, with p_2 as input, requires two units of work.

(2) The subsequent execution of FINDONE, with $q = AA$ as input, requires two units of work.

(3) The execution of FINDONE, with p_1 as input, requires three units of work.

(4) The subsequent execution of FINDONE, WITH $q = AAA$ as input, requires three units of work.

Therefore the total work units done by ADD, using the order p_2 followed by p_1 , is ten.

8. METHODS OF SATISFYING RULE 1

Fortunately RULE 1 is not as formidable as one might think and it can be satisfied in several simple ways. A few of these methods will now be discussed.

Undoubtedly the easiest way to comply is to append to each book a unique sequence of symbols. There are several ways in which this sequence could be obtained. The sequence could be book numbers that are generated by the BUILD INDEX algorithm. It could also be the start address, in the text, of the book or some combination of book number and the book start address. For example, in some situations it might be useful to chain the text together by inserting pointers to the preceding book and the succeeding book of a given book.

In satisfying RULE 1 by any of the above methods, every end includes the identifier of the book of which it is a right part. It is also clear that all ends which are right parts of the same book have different lengths and different starting addresses. Hence they generate unique ends.

There is another method of compliance. It is called the reverse library technique. Because of its complexity, it is not described here, but the reader is referred to either [4] or [5]. Its use is only recommended when none of the above methods can realistically be applied. In practice the librarian should look for some existing attribute of his library which would allow satisfaction of RULE 1.

9. EXTENDING PATRICIA TO PATRICIA-II

For efficient operation of PATRICIA, we must have fast access to the library index, especially to the twin and height indexes. In order to conform to the requirement of fast access, the library index should be stored in the computer's random access memory. Each start requires five storage locations in the index, two for twin, two for height, and one for start entries. Thus, if we assume our machine has r words of random access memory and sufficient external bulk storage to hold the library text, PATRICIA can efficiently handle a library with approximately $r/5$ starts. For modern day computers, $r/5$ is on the order of a few thousand.

We will address the problem of what to do when the random access memory of the host machine has been exhausted.

Some obvious solutions are as follows: first reduce the number of starts and reindex the entire library; secondly use a larger machine; and thirdly divide the library into sub-libraries.

Obviously the first two alternatives may not be feasible. The third alternative also has drawbacks. For example, to find all occurrences of a key, each sublibrary must be searched. The librarian must also decide in which sublibraries to place a given book.

We would like to extend PATRICIA in a manner which would retain as many of its efficiencies as possible. This is the subject of the next few paragraphs.

PATRICIA-II extends PATRICIA by building an index in a more general way. The index of PATRICIA-II consists of a lower index and an upper index. The lower index is small and is kept in core, while the upper index is large and is stored on disk or some other suitable bulk storage device. The upper index is subdivided into pages, which are read into core one at a time as needed. The lower index makes one reference to each page of the upper index. Each individual upper page refers only to itself and to the text.

PATRICIA-II makes a few more references to the disk than does the standard PATRICIA, but can handle a much larger number of starts. If we assume that the text is also stored on the disk, then PATRICIA-II is only slightly slower than PATRICIA. The difference in speed is a consequence of the additional references to index on disk made by PATRICIA-II. The number of references to the text by PATRICIA or PATRICIA-II is the same. The necessary text references are easy to calculate. If we are in the index-building phase, it requires $n + 1$ disk references to add a book with n starts. It requires one reference to add the book to the existing text and one reference to process each of the ends resulting from the starts. If we are operating in the query phase, and a key has $n \leq 1$ occurrences in the text, one reference to the disk is required. If a key has $n \geq 2$ occurrences, then $n + 1$ disk requests are required to retrieve all occurrences.

When PATRICIA-II is required to form a new upper page, which is seldom necessary, one disk reference is needed. If we are operating in the query phase, an average of one additional disk reference is required for the first occurrence of a key. This request is necessary to bring the correct upper page into core. Each additional occurrence of the key requires zero additional disk requests.

In order to better understand the PATRICIA-II concept, we will consider a simple example. Assume that PATRICIA has built a library that consists of sixteen ends and a corresponding index. Let us also assume that the binary tree representation of the index is that shown in Figure 9. Since we assumed the existence of sixteen ends and since the number of ends must be one more than the number of branches, the tree has thirty-one nodes. In order to proceed on to PATRICIA-II, let us make the stipulation that all nodes having more than five nodes above them are in the lower page, page zero. By examining the tree in Figure 10, we see that the lower page is enclosed by a dotted line. The upper pages are enclosed by solid lines. The lower page is core-resident and the upper pages are brought into core, one at a time, only when needed.

The author has implemented PATRICIA-II in order to demonstrate that the necessary algorithms work. A detailed discussion of this implementation is found in Section 15.

Figure 9.

The binary tree representation of an in-core index generated by PATRICIA.

Figure 10.

A paged index generated by PATRICIA-II
 (Solid lines enclose upper pages;
 dotted line encloses lower page.)

10. DEFINITION AND EXPLANATION OF THE BASIC VOCABULARY OF PATRICIA-II

For the most part, the vocabulary and definitions for PATRICIA-II are the same as those described in Section 3 for PATRICIA. Phrases which are doubly right extendible, i.e., by A and by B, are called branches. As in PATRICIA, branches are assigned even numbers. L-phrases which are right inextendible are assigned odd numbers and are called ends. Every end, except E_1 , has a stem which is a branch.

In PATRICIA-II it is necessary to classify ends and branches as being upper or lower. The lower ones occur in the lower index and the upper ones occur in the upper index.

A positive odd integer, N , plays a crucial role in the organization of the index. N is the number of nodes allowed in the binary tree which describes a given upper page of index. N is called the upper page size. Each upper page has another positive odd integer associated with it, called the page number. Because upper page numbering is a function of the PATRICIA-II index building scheme it is discussed in Section 12.1 and 12.2. Throughout the sequel, upper pages will be denoted by P_i for some positive odd i and the lower page by P_0 .

The contents of a given upper page consists of an index, as was described in Sections 3 and 4. It includes a start index, a twin index, and a height index. As in PATRICIA, the start index has an entry for each odd chain number, which points to the unique occurrence in the text of the phrases

in the corresponding odd chain. As in PATRICIA, each of the other two indexes has one entry for each chain number. Because of the paging scheme it is necessary that the index of an upper page contain a few minor additions which are described in Section 12.2.

The contents of the lower page are slightly different from what might be expected since there is no start index associated with it. Further discussion of the lower and upper page indexes is found in Sections 12.1 and 12.2.

At any given time there are only two pages in core, the lower page and an upper page P_i . The upper page, P_i , which is in core will be called the active upper page.

11. DETECTING THE PRESENCE OF A PHRASE AND FINDING ITS PROPER OCCURRENCES WITH PATRICIA-II

As in PATRICIA, we need to be able to find one proper occurrence of a phrase p . After having found one, we need to be able to find all proper occurrences. Our old friends, FINDONE and FINDALL, are able, with a little help, to do the job. The help comes from a new algorithm called LINK which is explained as follows.

11.1 The LINK Algorithm

This is an algorithm which is given an upper page number, P_i . Its task is to see what the current active upper page number, P_j , is. If $P_i \neq P_j$, then LINK must write page P_j out to the disk, then read page P_i into the core from the disk. If $P_i = P_j$ then the required upper page is the active upper page, and LINK merely returns to the calling routine. A flowchart of LINK is given in Figure 11.

11.2 The FINDONE Algorithm

This algorithm serves the same function in PATRICIA-II as it did in PATRICIA. The differences are minor ones necessitated by the paging scheme used in PATRICIA-II. The algorithm operates exactly as it did in PATRICIA. It begins execution in the lower page. When an odd chain number, c , is encountered in the lower page, FINDONE calls LINK. When LINK returns control to FINDONE, the active upper page is page P_c . FINDONE then runs to its usual termination. The output of FINDONE is a chain number, c_1 , which is the upper

Figure 11.
Flowchart for LINK.

chain number of the query phrase p , in the active upper page P_c . As in PATRICIA, if p is not in L , then c_1 is the chain number of a phrase p' . The phrase p' has in common with p a longest left part q , which is the longest left part of p in L .

If the execution of FINDONE on page zero outputs an even chain number, then it is necessary to read in more than one upper page in order to obtain the beginnings of all proper occurrences in L of p . This will be discussed further in the next section.

11.3 The FINDALL Algorithm

As in PATRICIA, FINDALL begins with an even chain number, c , and finds a sequence $c_1, c_2, \dots, c_k$ of odd chain numbers. A minor difference which arises here is the use of the above sequence. If c is the result of FINDONE operating on an upper page P_j , then FINDALL operates exactly as in PATRICIA. It references only the twin index of page P_j and returns only odd chain numbers from page P_j . As in PATRICIA, the chain numbers are such that the starts, S_{c_i} , are all the proper occurrences of the branch, $\tilde{B}_c$. The ends, E_{c_i} , are all the right inextendible right extensions of $\tilde{B}_c$.

In the event that c is the result of FINDONE operating on the lower page, page zero, then FINDALL returns a sequence, $\bar{c}_1, \bar{c}_2, \dots, \bar{c}_m$, of odd chain numbers from page zero. LINK then reads in upper pages one at a time, say P_j , for

$j = \bar{c}_1, \bar{c}_2, \dots, \bar{c}_m$. Each time an upper page is read into core, control is given to FINDALL if $c = TC(1)$ in the active upper page is even. Otherwise, a table lookup is done as described below. If control is given to FINDALL, it then returns all the odd chain numbers $c_1, c_2, \dots, c_k$ from the active upper page, P_j , in alphabetical order relative to the ends they define. Since P_j is the active upper page, the text addresses which correspond to the c_i 's can be found by performing k table lookups in the start index of page P_j . When this is completed we return to the sequence of $\bar{c}_j$'s for the next active upper page.

12. BUILDING THE LIBRARY INDEX FOR PATRICIA-II

As in PATRICIA, we need a procedure by which the library and corresponding index can be built. We are fortunate that the BUILD INDEX algorithm described in Section 7 is still applicable without modification. For a flowchart of BUILD INDEX, the reader should refer to Figure 6 in Section 7. The reason that the algorithm can be applied to PATRICIA-II without modification is that the ADD p routine does most of the work. Of all the algorithms used in PATRICIA-II that have counterparts in PATRICIA, it is the ADD algorithm which has undergone the most modifications. This is because of the paging scheme used in PATRICIA-II and the resulting lower index and upper index.

12.1 The Lower Index

As was mentioned earlier, the lower index is what makes up the lower page, P_0 , and is core-resident. It contains a twin index and a height index. In place of the start index we have what could be called a pseudo start index. We need the pseudo start index because when we add a phrase p to L it is possible that we can leave the FINDONE algorithm at EXIT 1 (see flowchart in Figure 4 of Section 5.1). In order to obtain a phrase p' , which has in common with p a longest left part q , we need the pseudo start index in page P_0 . In practice it is rarely used.

We cannot allow ends in P_0 , i.e. references to the text because this would defeat the purpose of page splitting by filling up page zero.

Odd entries in the twin index indicate lower ends, that is, references to upper pages. It is for this reason that upper pages have odd numbers which correspond to the lower end numbers.

12.2 The Upper Index

As was mentioned in Section 10, the contents of a given upper page consist of a twin index, a height index, and a start index. In order to simplify the ADD algorithm it is convenient to also designate an upper page as having a PATRICIA or a PATRICIA-II type index and to indicate an upper page as being full or not. An upper index is a PATRICIA type index if the upper page which contains it was not created by the upper page SPLIT algorithm. If an upper page, P_i , has a PATRICIA type index, then the ADD algorithm used in PATRICIA can be used on page P_i . An upper page is full if there is no room to add another end to it. In this case it is a candidate for splitting. It is also necessary to have a pointer to the root of the dead subtree of a page which has undergone splitting by the SPLIT algorithm. In order for the PRUN algorithm to operate efficiently on the dead subtree it is necessary to carry another pointer. This will be made clear after the discussion of the two algorithms SPLIT and PRUN which are discussed in Sections 13.1 and 13.2.

13. THE ADD ALGORITHM FOR PATRICIA-II

As in PATRICIA, ADD is an algorithm to modify the text and index of an existing library L to reflect the addition of a phrase p. However, the operations performed by the PATRICIA-II version of ADD are somewhat more complex. Because of this added complexity, two auxiliary algorithms are needed. They are called SPLIT and PRUN and will be discussed in Sections 13.1 and 13.2. They give ADD the capability of handling full upper pages and obtaining even branch numbers. In addition, ADD must be able to create a new upper page if necessary. A flowchart of the ADD algorithm is given in Figure 12.

In order to create a new upper page, the lower tree must be climbed until a lower end is encountered, that is, above the new lower branch. The pseudo start index is then used to obtain the text address of an end in L which shares with p a longest left part q. The active upper page is then written out to disk and a new active upper page is constructed. The new active upper page has the index that corresponds to the end p. The lower page index is then modified to reflect the formation of a new upper page.

13.1 The Upper Page SPLIT Algorithm

This is the algorithm which operates on a full upper page and produces two upper pages. The appearance of a new upper page in PATRICIA-II is analogous to the appearance of a new end in PATRICIA, i.e. page splitting brings a new branch into P_0 .

Figure 12.

Flowchart for PATRICIA-II ADD.

Figure 13.

A full upper page, P_5 , of index.

In order to gain some insight as to how we split an upper page into two pages, consider the binary tree in Figure 13. We shall assume the figure represents a full upper page, say P_5 , of index. Let us also assume that the next odd number available for an upper page is 17. If we remove the root of the binary tree in Figure 12, we obtain a left subtree and a right subtree. Let us designate the left subtree live and the right subtree dead and call these two subtrees the modified contents of P_5 . Now we create a new page, P_{17} , using these two subtrees and designate the right subtree live and the left subtree dead. By the live subtree it is meant that ADD will put new ends and branches in it. The dead subtree is used by ADD to obtain even branch numbers. The two upper pages, P_5 and P_{17} , where one is essentially the complement of the other, represent almost all of the information that was carried by the original contents of P_5 . The only information missing is that which was at the root of the original tree before splitting occurred. If we put this information in page zero, we will have all of the information that was carried in P_5 before splitting occurred.

When the lower end E_5 in page zero was encountered, P_5 was brought into core and was recognized as being full. This necessitated the splitting of P_5 . If we treat this lower end exactly as in PATRICIA (i.e., consider the act of upper page splitting as requiring the addition of a lower branch $\tilde{B}_{16}$ and a lower end E_{17}), we can put the missing information in the lower page.

The act of page splitting can be done in place. Let h be the height of the root of the binary tree in the page to be split. We examine h plus the first bit of p , which is the phrase being added. If it is an A, then the page which has the left subtree live should be in core after splitting. If it is a B, the other page should be in core when SPLIT is left. We need a rule by which upper page numbers can be systematically assigned when splitting occurs. The convention we will adopt is to let the page receiving the live left subtree keep the old page number and the page that receives the live right subtree take on the next available page number.

Since each upper page that has undergone splitting has a live tree and a dead tree, we must have a pointer to the root of the dead tree in order to access it. A pointer to the root of the live tree is not needed because, since it is live, the twin T_1 points to its root.

The operation of upper page splitting is a rare event in PATRICIA-II and is nonexistent in PATRICIA. Its number

of occurrences is a function of the upper page size N and the number of ends in a library L . The SPLIT algorithm splits the active upper page, say P_a , into two pages as follows:

- (1) The algorithm ADD, which is in the process of adding a phrase p to the library, has detected that P_a is full.
- (2) Determine, by examining the phrase p and the height of the root of the tree contained in P_a , which page (P_a or the new page), should be the active upper page when SPLIT is done.
- (3) Set up the page which should be written to disk. This includes turning off the full page indicator, setting up a pointer to the root of the dead subtree, and indicating the presence of a non-PATRICKIA type index.
- (4) Set up the new active upper page. This also includes turning off full page indicator, setting up a pointer to the dead tree, and indicating the presence of non-PATRICKIA type index.
- (5) Modify the contents of page zero to reflect the splitting of page P_a and the formation of a new upper page.

13.2 The Upper Page Dead Subtree PRUN Algorithm

In PATRICKIA, branch numbers were easy to obtain. When an even branch number was needed, it was easily generated. This is still the case when an upper page has a PATRICKIA type

index. The next available branch number is generated from the last end number that appeared.

In PATRICIA-II, the problem of obtaining an even branch number is more difficult since chain numbers in a given upper page must be used over and over again. The only chain numbers available for use are elements of the dead subtree and the root of the page before its last split. Because of this we need an algorithm by which we can extract even branch numbers from the dead subtree, and, after exhausting it, obtain the predecessor of the dead subtree root. The predecessor of the dead subtree root is the root of the original tree before its last split.

The algorithm which performs this operation is called PRUN [8]. PRUN must extract an even number for the new branch which is to be added to the live tree. The dead tree must remain a binary tree throughout the process. This requires that the removal of an even branch number from the dead tree must be accompanied by the removal of an odd end number since the number of ends must be one more than the number of branches. It is not required that the end number which is removed be the one that will be used with the branch number. It is perfectly all right to have some end numbers used in both the dead tree and the live tree and some in neither one. If an end number is in the dead tree, its only function is to indicate the end of a path in that tree. The algorithm for selecting a new branch number and pruning off an odd end number operates as follows:

- (1) For the branch, select the root of the dead tree.
If this number is odd, use the root of the page before the last occurrence of splitting. There is a possible pitfall here, but it may be avoided as outlined following step (4).
- (2) For the end, select the leftmost end of the left subtree of the dead tree. At this point the dead tree has been split into two parts, a left subtree with one end missing, and a right subtree.
- (3) Combine these two subtrees by attaching the root of the right subtree in place of the missing end of the left subtree. Also keep a pointer to the leftmost branch so that the next search for the leftmost end will begin here.
- (4) Make the root of the left subtree the root of the reconstructed dead tree.

As was mentioned in (1) above, if we use as our branch number the root of the page before the last occurrence of splitting, we could run into trouble. The reason for this is that if this even number is r , then its corresponding odd end number, $r+1$, could already be in use in the live tree. If so, it is not permissible to use it again. Recall that the stem of an end is defined for every end except E_1 . E_1 can be viewed as an orphan end number. We can determine if E_{r+1} is in use by going to the text and obtaining the corresponding phrase. If FINDONE is then executed, with this

phrase as input, the chain number $r+1$ will result if the end E_{r+1} is in use. If E_{r+1} is not in use, then we can use the end number $r+1$. If $r+1$ is in use, we must switch $r+1$ and l so that we can use $r+1$ as the end number corresponding to the branch number r .

At this point it might help to analyze the PRUN algorithm more closely. Directing our attention to the path in the dead subtree which leads from the root to the leftmost end, we find that each execution of PRUN conducts a search for the leftmost end of the left subtree. This end is then replaced by the root of the right subtree. We then designate the left successor of the old root as the new root of the dead tree. The old root is then extracted from the dead tree for use in the live tree. The only dynamic aspect of the algorithm is the search for the leftmost end.

The efficiency of the PRUN algorithm, for the case where the dead subtree is exhausted, is given in the following theorem:

Theorem

If the dead subtree was exhausted, then the average number of nodes examined per call to PRUN was 2.

Proof:

Each time the branch of the dead tree is extracted we are left with a left subtree and a right subtree. The search

for the leftmost end of the left subtree is then conducted. A pointer is then kept indicating the position of the leftmost end. The next path to be examined lies in the right subtree and none of its nodes have been examined so far. Since the algorithm uses the saved pointer to begin the next search and since the next path to be examined is in the right subtree it follows that each node is examined only once.

Each execution of PRUN will remove two nodes, the root and the leftmost end, from the dead tree. Since the tree is a binary tree, the number of nodes in it is odd and hence can be expressed by $2k+1$ for some k . Because two nodes are removed during each execution of PRUN, it will require $k+1$ executions to exhaust the tree. (The number of executions is either k or $k+1$ depending on whether we know if the tree has been exhausted or not when leaving the algorithm.)

Since each node is examined only once we can conclude that the new leftmost end is on the average only two nodes beyond the old leftmost branch.

Q.E.D.

For each new branch number produced, the following table lookups are required in the twin index: the old root, its left and right successors, the old leftmost branch, and the two steps required to find the leftmost end.

14. MULTIPLE KEYS AND USEFUL RETRIEVAL TECHNIQUES

As in PATRICIA, we can retrieve all books in which some intersection of keys is found [4]. PATRICIA-II handles multiple keys in the same way as PATRICIA. Assume a sequence of keys, $K_1, K_2, \dots, K_n$, is given and we are to retrieve all books which contain all of the above keys. When the first key, K_1 , is submitted as a query, FINDONE and FINDALL generate a list. This list contains the chain numbers of all books in which K_1 has a proper occurrence. The second key, K_2 , is then submitted as a query. This time a second list is generated, containing the chain numbers of all books in which both K_1 and K_2 have proper occurrences. After the last key, K_n , has been examined, the last list generated contains the chain numbers of the required books. In order to retrieve the books, the text is referenced in the usual manner through the start index. The storage required for the two lists generated by examining K_1 and K_2 is sufficient for the entire process. This is because the lists generated are monotonic nonincreasing by inclusion.

Since blanks are significant in PATRICIA and PATRICIA-II, we have a very useful retrieval technique [4]. The retrieval of all occurrences of a key Q without its right extensions is easily achieved. The user merely presents the key Q followed by a blank. Another useful technique, which has applications in sorting, is the retrieval of all occurrences of a key and its right extensions in alphabetical order.

15. DISCUSSION OF USEFUL PROGRAMMING TECHNIQUES AND COMPUTER IMPLEMENTATION

Since PATRICIA is a radix method of retrieving information, it is necessary to perform bit manipulation. This is necessary because FINDONE tests particular bits of a given phrase p . Moreover, ADD p must determine $\ell(q)$, the number of bits of agreement of p and p' . On machines which have floating-point normalize hardware, $\ell(q)$ can be determined very efficiently. This is done by obtaining the "exclusive or" of p and p' , determining the first nonzero word of it, and then normalizing.

In general it is also necessary to manipulate characters. This is due to the fact that in general character strings do not begin and end on a word boundary.

Since odd positive integers are assigned to starts and since $S_{n+1} = S_{n+2}$, for odd n , one cell can hold this information. Therefore, the start array should be half the size of the twin array or height array.

PATRICIA-II has been implemented on a CDC 6600 and 7600. It is written in a low level FORTRAN with the exception of two assembly level subroutines. These two subroutines are a left shift circular routine and a normalization routine.

The program has two character manipulation routines for fetching and storing characters. For better efficiency these two subroutines should be written in assembly language. They are in FORTRAN so that the task of transferring to a different target machine is made easier.

The program is highly subroutinized and contains ample comments. The basic routines FINDSTART, GETBOOK, and GETQUERY are distinct subroutines. They can easily be changed or replaced to accommodate the needs of the librarian. The reader is referred to the appendix for samples of input and output.

Very few difficulties should be encountered in transferring the code to any target machine which has some type of bulk storage device and a FORTRAN language implemented.

After implementing PATRICIA-II it is possible to suggest improvements and modifications. These suggestions are the product of empirical results obtained by indexing a particular library. They are also relevant only when the bulk storage device is a disk with movable heads. This is not a very big restriction since the movable head disk is still the primary bulk storage device.

The library used consisted of English text. The number of authors of the text was approximately twenty.

It is clear that an increase in the upper page size, N , is accompanied by a decrease in the frequency of upper page splitting. In order to reduce the number of times page splitting occurs one would maximize N . Unfortunately, this increases the quantity of data moved to and from the disk each time a new page is read into core. However, if N is too small, page splitting will increase, which will also result

in a larger number of disk accesses, and the number of nodes in the lower page will increase.

Empirical results are shown in Figure 14 through Figure 16 for the two phases, library indexing and querying. These results were obtained on a CDC 7600 utilizing the LASL CROS batch operating system. The library consisted of 10000 ends.

In most applications it is the user response time that should be minimized. This implies that the number of upper pages should be kept small.

The predominant factor in determining an optimum upper page size, N , is the operating system on the machine in question. The system buffer sizes, data buffering algorithm, and the rate at which the hardware transfers the data are the significant parameters. Since these vary with machines, the best procedure is to empirically determine the optimum page size for a given machine.

Some machines have what is called an extended memory in addition to main memory. Both the access time and the transfer rate are much faster than disk. If available, one could use the extended memory to store upper pages. When a new page is needed, extended memory would be examined first. If the required page was not there it could be read in from disk.

Figure 14.
Graph of upper page size versus real time
in seconds to query entire library.

Figure 15.

Graph of upper page size versus real time
in seconds to index entire library.

Figure 16.

Graph of real time to query
entire library versus number of upper pages.

The ideal environment in which to implement PATRICIA-II would be where the machine has, in addition to main memory, an extended memory, a large bulk storage device, and a graphical device which has a light pen or tablet feature. The graphical device would be very useful for selecting starts and submitting keys. The operating system should at least provide multiprogramming. Preferably it should be a time sharing system.

To give some indication of the efficiency of PATRICIA, compared to the binary search, the average number of branches visited on a retrieval key was computed for a varying number of starts. The empirical results are shown in Figure 17. The library used was the one described above. For this particular set of data PATRICIA is about half as efficient as the binary search. However, it must be remembered that the binary search applies only to an alphabetically ordered list, while PATRICIA applies to any set of starts in any text.

Figure 17.

Comparison of PATRICIA and the binary search.
(Solid line is binary search, dotted line is PATRICIA.)

16. APPLICATIONS

PATRICIA-II can be applied to the same type of problems as its predecessor. The main difference is that PATRICIA-II can handle much larger libraries.

In the area of text editing we can use PATRICIA-II to find all occurrences of a given string in the text. Another use is in the sorting on keys of variable length records. In fact this is one advantage to using the algorithm. Usually it is necessary to sort the output so that the results are presented in an organized manner.

The problem of determining which text addresses should be start addresses is not as formidable as one might first think. This is because the text is considered to be a linear string of bits, and starts are bit addresses in this string.

For example, assume that the data base to be indexed is on some medium suitable for use on a computer. This could be paper on which an optical character recognizer operates, punched cards, magnetic tape, or some other means.

The librarian should look for some attribute of the data which allows easy identification of starts such as fixed record length, a certain character, or possibly a sequence of characters.

SSSSS ADD THE FOLLOWING SET OF BOOKS TO THE LIBRARY

STARTS ARE DESIGNATED BY + AND STOPS BY +

+ZIP +87544 +CITY +LOS ALAMOS +STREET +TODD LP +NUM 559 +NAME +CLARK,J
+L. +PHONE +672-3418+

+ZIP +87501 +CITY +SANTA FE +STREET +SAN JOSE +NUM 7131 +NAME +SNIDER,
D.M. +PHONE +389-0794+

+ZIP +87727 +CITY +LOS LUNAS +STREET +MAIN +NUM 2782 +NAME +WINKEL,R.M.
+PHONE +NONE+

+ZIP +87743 +CITY +ROY +STREET +P.O. BOX +NUM 1663 +NAME +LITTLE,C.
+PHONE +210-8734+

+ZIP +87501 +CITY +SANTA FE +STREET +SAN MARCUS +NUM 235 +NAME +SLOAN,
D.J. +PHONE +289-7569+

+ZIP +87544 +CITY +LOS ALAMOS +STREET +WOODLAND +NUM 3589 +NAME +ORR,S
+M. +PHONE +662-3829+

+ZIP +87501 +CITY +SANTA FE +STREET +SAN LUIS +NUM 1317 +NAME +SNIDER,
M.D. +PHONE +389-0497+

+ZIP +87501 +CITY +SANTA FE +STREET +CAMINO DEL GUSTO +NUM 8292 +NAME
+SNYDER,F.M. +PHONE +389-0007+

THE LIBRARY NOW CONSISTS OF 6 BOOKS

Sample Input/Output

APPENDIX

SSSSSSSSSSSSSSSSSSSS SINGLE KEYS ARE BEING PROCESSED SSSSSSSSSSSSSSSSSSS

1ZIP 87544

THE PHRASE IS

ZIP 87544 CITY LOS ALAMOS STREET TODD LP NUM 559 NAME CLARK,J.L. PHONE 672-3418

..... THE ABOVE PHRASE IS BOOK NUMBER 3

THE PHRASE IS

ZIP 87544 CITY LOS ALAMOS STREET WOODLAND NUM 3589 NAME ORR,S.R. PHONE 662-5829

..... THE ABOVE PHRASE IS BOOK NUMBER 8

SSSSSSSSSSSSSSSSSSSS SINGLE KEYS ARE BEING PROCESSED SSSSSSSSSSSSSSSSSSS

187544+

THE PHRASE IS

87544 CITY LOS ALAMOS STREET TODD LP NUM 559 NAME CLARK,J.L. PHONE 672-3418

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 3 WHICH IS PRINTED BELOW

ZIP 87544 CITY LOS ALAMOS STREET TODD LP NUM 559 NAME CLARK,J.L. PHONE 672-3418

THE PHRASE IS

87544 CITY LOS ALAMOS STREET WOODLAND NUM 3589 NAME ORR,S.R. PHONE 662-5829

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 8 WHICH IS PRINTED BELOW

ZIP 87544 CITY LOS ALAMOS STREET WOODLAND NUM 3589 NAME ORR,S.R. PHONE 662-5829

SSSSSSSSSSSSSSSSSSSS SINGLE KEYS ARE BEING PROCESSED SSSSSSSSSSSSSSSSSSS

1STREET TODD LP+

THE PHRASE IS
STREET TODD LP NUM 559 NAME CLARK,J.L. PHONE 672-3418

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 3 WHICH IS PRINTED BELOW

ZIP 67544 CITY LOS ALAMOS STREET TODD LP NUM 559 NAME CLARK,J.L. PHONE 672-3418

SSSSSSSSSSSSSSSSSSSS SINGLE KEYS ARE BEING PROCESSED SSSSSSSSSSSSSSSSSSSSS

1P.O. BOX

THE PHRASE IS
P.O. BOX NUM 1663 NAME LITTLE.C. PHONE 210-8734

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 6 WHICH IS PRINTED BELOW

ZIP 87743 CITY ROY STREET P.O. BOX NUM 1663 NAME LITTLE.C. PHONE 210-8734

SSSSSSSSSSSSSSSSSSSS SINGLE KEYS ARE BEING PROCESSED SSSSSSSSSSSSSSSSSSSSS

1PHONE

THE PHRASE IS
PHONE NONE

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 8 WHICH IS PRINTED BELOW

ZIP 87727 CITY LOS LUNAS STREET MAIN NUM 2782 NAME WINKEL,B.H. PHONE NONE

THE PHRASE IS
PHONE 210-8734

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 6 WHICH IS PRINTED BELOW

ZIP 87743 CITY ROY STREET P.O. BOX NUM 1663 NAME LITTLE.C. PHONE 210-8734

THE PHRASE IS
PHONE 289-7569

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 7 WHICH IS PRINTED BELOW

ZIP 87501 CITY SANTA FE STREET SAN MARCUS NUM 235 NAME SLOAN,D.J. PHONE 289-7549

THE PHRASE IS
PHONE 389-0007

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 10 WHICH IS PRINTED BELOW

ZIP 87501 CITY SANTA FE STREET CAMINO DEL GUSTO NUM 8292 NAME SNYDER,F.M. PHONE 389-0007

THE PHRASE IS
PHONE 389-0497

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 9 WHICH IS PRINTED BELOW

ZIP 87501 CITY SANTA FE STREET SAN LUIS NUM 1317 NAME SNIDER,H.D. PHONE 389-7497

THE PHRASE IS
PHONE 389-0794

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 4 WHICH IS PRINTED BELOW

ZIP 87501 CITY SANTA FE STREET SAN JOSE NUM 7131 NAME SNIDER,D.H. PHONE 389-7794

THE PHRASE IS
PHONE 662-5829

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 8 WHICH IS PRINTED BELOW

ZIP 87544 CITY LOS ALAMOS STREET WOODLAND NUM 3589 NAME ORR,S.R. PHONE 662-5829

THE PHRASE IS
PHONE 672-3418

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 3 WHICH IS PRINTED BELOW

ZIP 87544 CITY LOS ALAMOS STREET TODD [P NUM 559 NAME CLARK,J.L. PHONE 672-3418

SSSSSSSSSSSSSSSSSSSS SINGLE KEYS ARE BEING PROCESSED SSSSSSSSSSSSSSSSSSSSS

1389-4
THE PHRASE IS
389-0007

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 10 WHICH IS PRINTED BELOW

ZIP 87501 CITY SANTA FE STREET CAMINO DEL GUSTO NUM 8292 NAME SNYDER.F.H. PHONE 389-0007

THE PHRASE IS
389-0497

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 9 WHICH IS PRINTED BELOW

ZIP 87501 CITY SANTA FE STREET SAN LUIS NUM 1317 NAME SNIDER.H.D. PHONE 389-0497

THE PHRASE IS
389-0794

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 4 WHICH IS PRINTED BELOW

ZIP 87501 CITY SANTA FE STREET SAN JOSE NUM 7131 NAME SNIDER.D.H. PHONE 389-1794

SSSSSSSSSSSSSSSSSSSS SINGLE KEYS ARE BEING PROCESSED SSSSSSSSSSSSSSSSSSS

NAME *

THE PHRASE IS
NAME CLARK.J.L. PHONE 672-3418

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 3 WHICH IS PRINTED BELOW

ZIP 87544 CITY LOS ALAMOS STREET TODD LP NUM 559 NAME CLARK.J.L. PHONE 672-3418

THE PHRASE IS
NAME LITTLE.C. PHONE 210-8734

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 6 WHICH IS PRINTED BELOW

ZIP 87743 CITY RDY STREET P.O. BOX NUM 1663 NAME LITTLE.C. PHONE 210-8734

THE PHRASE IS
NAME ORR.S.R. PHONE 662-5829

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 8 WHICH IS PRINTED BELOW

ZIP 87544 CITY LOS ALAMOS STREET WOODLAND NUM 3589 NAME ORR,S.R. PHONE 662-5429

THE PHRASE IS
NAME SLOAN,D.J. PHONE 289-7569

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 7 WHICH IS PRINTED BELOW

ZIP 87501 CITY SANTA FE STREET SAN MARCUS NUM 235 NAME SLOAN,D.J. PHONE 289-7549

THE PHRASE IS
NAME SNIDER,D.M. PHONE 389-0794

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 4 WHICH IS PRINTED BELOW

ZIP 87501 CITY SANTA FE STREET SAN JOSE NUM 7131 NAME SNIDER,D.M. PHONE 389-4794

THE PHRASE IS
NAME SNIDER,M.D. PHONE 389-0497

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 9 WHICH IS PRINTED BELOW

ZIP 87501 CITY SANTA FE STREET SAN LUIS NUM 1317 NAME SNIDER,M.D. PHONE 389-4497

THE PHRASE IS
NAME SNYDER,F.M. PHONE 389-0007

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 10 WHICH IS PRINTED BELOW

ZIP 87501 CITY SANTA FE STREET CAMINO DEL GUSTO NUM 8292 NAME SNYDER,F.M. PHONE 389-0007

THE PHRASE IS
NAME WINKEL,B.M. PHONE NONE

THE ABOVE PHRASE IS FOUND IN BOOK NUMBER 5 WHICH IS PRINTED BELOW

ZIP 87727 CITY LOS LUNAS STREET MAIN NUM 2782 NAME WINKEL,B.M. PHONE NONE

SSSSSSSSSSSSSSSSSS SINGLE KEYS ARE BEING PROCESSED SSSSSSSSSSSSSSSSSSS

22IP.

THE PHRASE IS
ZIP 87501 CITY SANTA FE STREET CAMINO DEL GUSTO NUM 8292 NAME SNYDER,F.M. PHONE 389-0007

..... THE ABOVE PHRASE IS BOOK NUMBER 10

THE PHRASE IS
ZIP 87501 CITY SANTA FE STREET SAN JOSE NUM 7131 NAME SNIDER,D.M. PHONE 389-0794

..... THE ABOVE PHRASE IS BOOK NUMBER 4

THE PHRASE IS
ZIP 87501 CITY SANTA FE STREET SAN LUIS NUM 1317 NAME SNIDER,M.D. PHONE 389-0497

..... THE ABOVE PHRASE IS BOOK NUMBER 9

THE PHRASE IS
ZIP 87501 CITY SANTA FE STREET SAN MARCUS NUM 235 NAME SLOAN,D.J. PHONE 289-7569

..... THE ABOVE PHRASE IS BOOK NUMBER 7

THE PHRASE IS
ZIP 87544 CITY LOS ALAMOS STREET TODD LP NUM 559 NAME CLARK,J.L. PHONE 672-3418

..... THE ABOVE PHRASE IS BOOK NUMBER 3

THE PHRASE IS
ZIP 87544 CITY LOS ALAMOS STREET WOODLAND NUM 3589 NAME ORR,S.R. PHONE 662-5829

..... THE ABOVE PHRASE IS BOOK NUMBER 8

THE PHRASE IS
ZIP 87727 CITY LOS LUNAS STREET MAIN NUM 2782 NAME HINKEL,M.M. PHONE NONE

..... THE ABOVE PHRASE IS BOOK NUMBER 5

THE PHRASE IS
ZIP 87743 CITY ROY STREET P.O. BOX -NUM 1663 NAME LITTLE.C. PHONE 210-8734

..... THE ABOVE PHRASE IS BOOK NUMBER 6

SSSSSSSSSSSSSSSSSSSSSS MULTIPLE KEYS ARE BEING PROCESSED SSSSSSSSSSSSSSSSSSSSSSS

FIND ALL BOOKS WHICH HAVE THE FOLLOWING KEYS IN COMMON

1LOS ALAMOS+
2TODD LP+

1 BOOKS SATISFY THE INTERSECTION

BELOW ARE THE BOOKS WHICH SATISFY THE INTERSECTION

.ZIP 87544 CITY LOS ALAMOS STREET TODD LP NUM 559 NAME CLARK,J.L. PHONE 672-3418

SSSSSSSSSSSSSSSSSSSS MULTIPLE KEYS ARE BEING PROCESSED SSSSSSSSSSSSSSSSSSSSS

FIND ALL BOOKS WHICH HAVE THE FOLLOWING KEYS IN COMMON

1SANTA FE+
2STREET S+

3 BOOKS SATISFY THE INTERSECTION

BELOW ARE THE BOOKS WHICH SATISFY THE INTERSECTION

ZIP 87501 CITY SANTA FE STREET SAN JOSE NUM 7131 NAME SNIDER,D.H. PHONE 389-0794

ZIP 87501 CITY SANTA FE STREET SAN LUIS NUM 1317 NAME SNIDER,H.D. PHONE 389-0497

ZIP 87501 CITY SANTA FE STREET SAN MARCUS NUM 235 NAME SLOAN,D.J. PHONE 289-7569

SSSSSSSSSSSSSSSSSSSSSE MULTIPLE KEYS ARE BEING PROCESSED SSSSSSSSSSSSSSSSSSSSSSS

FIND ALL BOOKS WHICH HAVE THE FOLLOWING KEYS IN COMMON

107501+
1STREET SAN+
1CITY 3+
2NAME SN+

2 BOOKS SATISFY THE INTERSECTION

BELOW ARE THE BOOKS WHICH SATISFY THE INTERSECTION

ZIP 07501 CITY SANTA FE STREET SAN JOSE NUM 7137 NAME SNIDER,D.M. PHONE 389-0794

ZIP 07501 CITY SANTA FE STREET SAN LUIS NUM 1317 NAME SNIDER,M.D. PHONE 389-0497

SUBROUTINE QUERY ENCOUNTERED A STOP CONTROL CARD

ACKNOWLEDGMENTS

The research reported in this dissertation was done while I was employed by the Los Alamos Scientific Laboratory. Support was provided by the U. S. Atomic Energy Commission under contract with the Laboratory. I am grateful to both the Commission and the Laboratory for their assistance.

I particularly wish to express my deep appreciation to Professor Donald R. Morrison of the University of New Mexico for his invaluable advice, guidance, and encouragement while supervising this research. I am also indebted to the other members of my committee, Professors Edgar J. Gilbert and Lawrence F. Shampine, for their personal and professional interest.

I offer special thanks for the help and ideas of my friends and colleagues at the Laboratory, in particular Messrs. Ray Davenport and Karl J. Melendez and Dr. Thomas L. Jordan.

REFERENCES

1. Belzer, Jack, Comments on "The Pros and Cons of a Special IR Language," comm. ACM 4, 9 (September 1961), p. 377.
2. Gear, C. William, Computer Organization and Programming, McGraw-Hill Book Co., Inc., New York, 1969.
3. Knuth, Donald E., The Art of Computer Programming, Vol. 1, Addison-Wesley Publishing Co., 1968.
4. Morrison, Donald R., "PATRICIA-Practical Algorithm to Retrieve Information Coded in Alphanumeric," Report SC-RR-67-734, Sandia Corp., Albuquerque, N. Mex., Oct. 1967.
5. Morrison, Donald R., "PATRICIA-Practical Algorithm to Retrieve Information Coded in Alphanumeric," J. ACM 15, 4 (October 1968), pp. 514-534.
6. Morrison, Donald R., "A Library Automaton," unpublished notes.
7. Morrison, Donald R., "Binatural Numbers," unpublished notes.
8. Morrison, Donald R., private communication.
9. Salton, Gerald, Automatic Information Organization and Retrieval, McGraw-Hill Book Co., Inc., New York, 1968.
10. Sammet, Jean E., Comments on "The Pros and Cons of a Special IR Language," comm. ACM 5, 1 (January 1962), p. 8.
11. Simmons, George F., Introduction to Topology and Modern Analysis, McGraw-Hill Book Co., Inc., New York, 1963.