PATRICIA—Practical Algorithm To Retrieve Information Coded in Alphanumeric

DONALD R. MORRISON

Sandia Laboratory, Albuquerque, New Mexico*

ABSTRACT. PATRICIA is an algorithm which provides a flexible means of storing, indexing, and retrieving information in a large file, which is economical of index space and of reindexing time. It does not require rearrangement of text or index as new material is added. It requires a minimum restriction of format of text and of keys; it is extremely flexible in the variety of keys it will respond to. It retrieves information in response to keys furnished by the user with a quantity of computation which has a bound which depends linearly on the length of keys and the number of their proper occurrences and is otherwise independent of the size of the library. It has been implemented in several variations as FORTRAN programs for the CDC-3600, utilizing disk file storage of text. It has been applied to several large information-retrieval problems and will be applied to others.

KEY WORDS AND PHRASES: indexing, information retrieval, keys

CR CATEGORIES: 3.7

1. *Context and Purpose*

Libraries and files are constructed to store information. Examples include personnel files, telephone directories, part lists, dictionaries (scientific or technical), and general libraries. The *user* of a large library or file approaches the library in search of information. We call a piece of information sought by a user a *target*. Typically, a target is a book, a poem, a chapter, an article, a definition, a theorem, a biography, a part description, an employee's record, or the name, address, and telephone number of a person, a customer, or a supplier. When he approaches the library in search of information, the user has in his possession something which we call a *key*, which, he hopes, will enable him, or help him, to recognize the target if and when he finds it. The key is a small piece of the target—typically, the title or a part of the title, the author's name, a prominent line, an important word, a part name or serial number, a person's name or address or telephone number, or the word whose definition is sought, or the hypothesis or conclusion of the target theorem.

A user does not want to read the whole library to find a target. Anticipating this desire, *librarians* construct indexes, catalogs, and other devices to accelerate the discovery of targets for which they anticipate that keys may be presented. From the users' point of view, it is desirable that the index be approachable with keys which are not unduly restricted or artificial in format, and that the index itself not require interminable scanning before it reveals the location of a target. From the librarian's point of view, it is desirable that an index serve the needs of the users; that it be economical of space and of the time of the librarian; that it be

* Computer Science, Division 5256. This work was supported by the US Atomic Energy Commission.

easily modified to reflect additions to or deletions from the library; and that such additions or deletions require a minimum of relocation of remaining items in the library and the index. Ideally, as the library grows, the quantity of effort required to find all occurrences of a particular key, or to add a new target and to modify the index accordingly, should remain bounded. The bound should depend only on the key or target and the number of its occurrences in the library and be otherwise independent of the size of the library.

The requirements outlined above are probably mutually contradictory when users and librarians are people and the library is coded in an unrestricted variety of alphabets, languages, and formats. When, however, the users and librarians are high-speed binary digital computers, and the library is coded in a binary alphabet, techniques become available which achieve all of the above requirements.

PATRICIA is a system for constructing an index for a binary coded library, for using the index to retrieve targets for which keys are presented, and for modifying the index to reflect changes in the library. PATRICIA achieves these objectives with quantities of memory and computation which appear to be near minimal for the problem defined. In particular, the index which PATRICIA defines includes no keys or text, only numbers which designate locations in the text or index. The amount of computation required to find one occurrence of a key, if there is one, or to find that there is none, has a bound which depends linearly on the length of the key and is independent of the size of the library. The quantity of computation required to find all occurrences of a key after one occurrence has been found depends linearly on the number of such occurrences and is independent of the size of the library. The computational effort involved in adding a new item to the library and adjusting the index to reflect its presence consists of copying the new item into the text, determining how much of it is already present, as outlined above, adding five numbers, and changing one in the previously existing index. The keys to which PATRICIA responds occur in the text, not in the index. They occur, therefore, only in their natural form, as pieces of the text, and are not restricted in format or length, except to the extent that the stoarge available for text is so restricted. The text is likewise unrestricted in format. Items in the text need not be arranged in any special order, except insofar as semantic continuity requires it; they can be stored in the order they are received. New additions will not necessitate relocation of old ones, and items can be of uniform or variable length. Keys and targets need not be in one-to-one correspondence; one target may be retrievable by as many keys as the librarian chooses to identify in it, and one key may retrieve as many targets as it has occurrences so identified.

2. *Motivations*

The referee and other perceptive readers have reacted to Sections 3–5 with the thought that there must be other ways to implement PATRICIA—ways which are less restrictive and easier to understand. There are. Some of them were stages in the evolution which led to the present version. It may be helpful to the reader to know some of the reasoning which led to the present version.

PATRICIA evolved from "A Library Automaton" [3] and other studies. Its evolution is continuing in the current development of PATRICIA II, an extension of PATRICIA, which makes better use of external storage of the index. Early in this evolution it was decided that the alphabet should be restricted to a binary one. A theorem which strongly influenced this decision is one which, in another form, is due to Euler. The theorem states that if the alphabet is binary, then the number of branches is exactly one less than the number of ends. Corollaries state that as the library grows, each new end brings into the library with it exactly one new branch, and each branch has exactly two exits. These facts are very useful in the allocation of storage for the index. They imply that the total storage required is completely determined by the number of ends, and all of the storage required will actually be used.

If the alphabet is not binary, then none of these results hold. Some new ends bring with them new branches; others create new exits to old branches. The number of branches is not predictable from the number of ends, and the number of exits from branches varies from branch to branch. All of these facts pose dilemmas for the programmer when he tries to allocate storage for the index. If he allocates storage for the maximum number of branches possible with a given number of ends, and for the maximum number of exits for each branch, then much precious fast-access storage space is unused. If he tries to overcome this difficulty by a threaded list, or some equally sophisticated scheme, it will still cost him precious fast-access storage and execution time as well. Another complication arises in the innermost loops of the algorithms in that many instruction sequences which are a single path in the binary case have branches to cover the several possibilities which arise in the more general case. This is also costly in fast-access storage.

Experience with PATRICIA and its predecessors on a variety of applications which are thought to be reasonably typical of those to which it can be applied indicates that the TWIN table, especially that part of it which corresponds to chains of low height, is accessed with much greater frequency than any other part of PATRICIA's data base. It is therefore desirable, in the interest of efficiency, that all of the TWIN table, or as much as possible of its lower part, be stored in the fast-access memory. Fast access to the HEIGHT table is only a little less essential, and to the START table and the TEXT, much less. Many of the choices made in formulating the present version of PATRICIA were based on the desire to get as many entries as possible of TWIN and HEIGHT tables into the core. The number of entries in these tables which can be packed into core determines, more than any other parameter, the flexibility and efficiency of PATRICIA.

The numbering scheme for ends and branches, which assigns odd numbers to ends and even numbers to branches, is not essential, but it does have several advantages over its alternatives. It is necessary, somehow, to distinguish between ends and branches. This can be done by storing an explicit Boolean variable or by making the dichotomy implicit in some feature of the storage, such as the numbering of ends and branches. If numbers in one class are to be assigned to ends, and in another class to branches, then there is some advantage in choosing the two classes of numbers in such a way that, like the ends and branches, they come naturally in pairs, consisting of one from each class. Even and odd positive integers have that property. So do positive and negative integers. Even and odd positive integers are preferred to positive and negative integers, because they can be packed and unpacked more easily, several to a word, if that should be necessary to conserve core storage space, and because they can be utilized more readily as addresses and as indexes in arrays.

The START table is an indirect addressing scheme, linking the odd-numbered chains with the text. The indirect scheme was preferred to direct addressing for the following reasons. Since the text is much larger and more heterogeneous than the index, the text addresses range over a much wider spectrum than the chain numbers, and each one requires more storage space. What is more, the text will probably be stored externally, and the format of text addresses will be dictated by the external hardware. By utilizing the START table as a link between the TWIN table and the text, the problems of internal file management for the TWIN table and external file management for the text become almost completely independent. They intersect only in the relatively simple matter of designing the START table and instructions for reading and writing in it.

The updating algorithm, ADD p, may seem unduly complicated. The most difficult part of ADD p's task is that of finding where to branch from. Once that is accomplished, the process of actually modifying the text and index is trivial. It would be conceptually easier to find where to branch from by scanning the index. This would, however, destroy PATRICIA's greatest asset: its ability to update a growing library in a number of computational steps which does not grow linearly with the length of the previously accumulated index. To retain this advantage, ADD p utilizes FINDONE, a scheme which is moderately hard to understand, but far more efficient in execution than any scanning technique, when the index is large. FINDONE looks at branches only, ignoring intervening nonbranches until the end of its search, at which time it checks all of these simultaneously. If FINDONE looked at each branch and nonbranch in order, then it would be intermittently looking at text and index. If, as is assumed, the index is readily accessible, and the text is less so, then the scheme which takes only one final look at the text is much more efficient.

In summary, the choices which were made in formulating the present version of PATRICIA represent the author's estimate of the best compromise among the desiderata of simplicity, lucidity, and efficiency, in the environment of existing hardware and anticipated applications. In that environment, text is stored in slow-access memory and as much as possible of the index in fast-access memory.

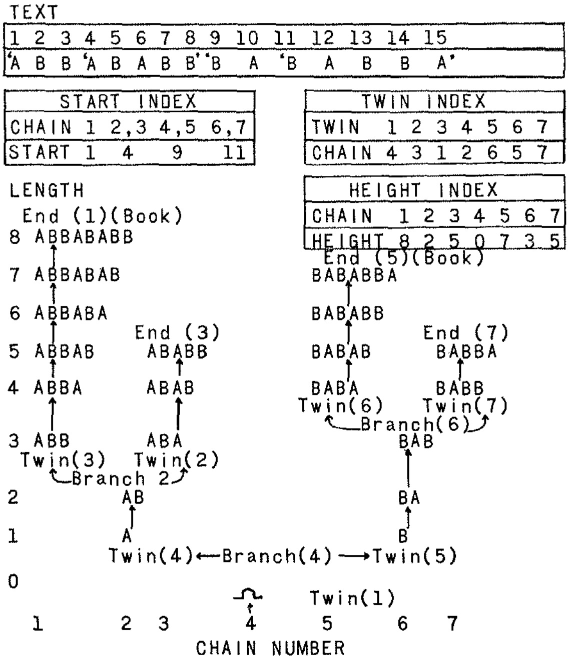
3. *START, STOP, END, L-PHRASE, BRANCH, TWIN, and CHAIN*

These seven words form the basic vocabulary of PATRICIA, and each is used here with a highly specialized meaning. To emphasize the specialized meaning, these words are set in capitals throughout this section (only). For examples of the terms here defined, see Figure 1. The TEXT of a RIGHT LIBRARY[1] is a finite sequence of bits (binary symbols); the binary symbols are here denoted by A and B.[2] The symbol Ω denotes the *null phrase* or phrase of length zero. Each position in the TEXT is identified by a number called its *address* and is occupied by an A or a B. Text addresses are consecutive positive integers, increasing as the text is read from left to right.

Certain text addresses are designated as STARTs and others as STOPs. These

[1] It is so called because it is read from left to right.

[2] A and B are preferred to the more conventional 0 and 1, because numerals appear in the index and it is desirable to maintain a distinction between symbols which appear in the text and numerals which appear in the index.



TEXT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
‘A	B	B	‘A	B	A	B	B’	‘B	A	‘B	A	B	B	A’

START INDEX				
CHAIN	1	2,3	4,5	6,7
START	1	4	9	11

TWIN INDEX							
TWIN	1	2	3	4	5	6	7
CHAIN	4	3	1	2	6	5	7

HEIGHT INDEX							
CHAIN	1	2	3	4	5	6	7
HEIGHT	8	2	5	0	7	3	5

Fig. 1. Representations of a right library

designations need not be marked in the TEXT itself; they are noted by recording the designated addresses in the index. A START occurs at the beginning of each occurrence of a key, which, in the librarian's judgment, should be pointed out to users who enter with that key. Such an occurrence is called a PROPER OCCURRENCE. The librarian's judgment is implemented as a subroutine, FINDSTART, upon which the main program, PATRICIA, calls to build an index. Typically, a START is indicated at the beginning of each item of information likely to be a target, that is, each book, paragraph, personnel record, etc. At the librarian's discretion, a START may also be designated at selected places interior to each likely target—at the beginning of a name, an address, a telephone number, a prominent word, or any part of the target likely to be used as a key to the target. If the librarian so directs, a START may be designated at the beginning of every word. The quantity of storage and computation required to construct the index is roughly proportional to the number of places in the text which are designated by the librarian as STARTs.

In the example of Figure 1, for the guidance of the reader, a START is marked by a single opening quotation mark: ‘.

A STOP is a place in the text which is at the right end of an item which is likely to be a target—a place which separates material to its right from unrelated material to its left. It corresponds roughly to what is called, in conventional terminology, an end of record or end of book. In Figure 1, a STOP is indicated by a single closing quotation mark: ’.

PATRICIA seeks only PROPER OCCURRENCEs of phrases, that is, occurrences which *begin at a* START and *end not later than the next* STOP. Any phrase which has at least one PROPER OCCURRENCE is called an L-PHRASE. The set of all L-PHRASEs is called L, or the LIBRARY. An L-PHRASE which begins at a START and ends *at* the next STOP is called an END. Every left part of an END is an L-PHRASE, and conversely, every L-PHRASE is a left part of at least one END. The L-PHRASEs in Figure 1 are the phrases to which upward arrows (↑) point. The listing of L-PHRASEs is shown in Figure 1 only to assist the reader in identifying and classifying L-PHRASEs. No such listing appears in the computer memory. Only the TEXT and the START, TWIN, and HEIGHT indexes appear in the computer memory.

> Rule 1. Each END has exactly one PROPER OCCURRENCE in the TEXT, beginning at one and only one START and ending at the next STOP.

Rule 1 does not restrict or limit in any way the kind of information which can be stored; it can be complied with in any one of several simple ways. These are described in Sections 8 and 9.

An END is a BOOK if the START at which it begins is *leftmost* among the STARTs corresponding to its stop; that is, it is not a right part of another END. In the example shown in Figure 1, ABBABABB, ABABB, BABABBA, and BABBA are ENDs; but only ABBABABB and BABABBA are BOOKs, because the other two ENDs are right parts of these. Every END is a right part of at least one BOOK.

If p is any L-PHRASE, then pA and pB are called *minimal right extensions of p*. Either, or both, or neither of pA and pB may be in L. If neither is in L, then p is an END. If both pA and pB are in L, then p is a BRANCH in L. This happens when pA is a left part of one END, and pB is a left part of another END. In the example of Figure 1, BAB is a BRANCH because its two minimal right extensions BABA and BABB are both left parts of ENDs: BABA is a left part of the END BABABBA, and BABB is a left part of the END BABBA. Similarly, AB and Ω are BRANCHes in L; their minimal right extensions ABA, ABB, A and B are all in L.

If p is a BRANCH in L, then its two minimal right extensions are each called a TWIN in L. In Figure 1, (A,B), (ABA,ABB), and (BABA,BABB) are TWIN L-phrases because Ω, AB, and BAB are BRANCHes. For reasons to appear, Ω is also classed as a TWIN, although it is not an extension of a BRANCH.

It is useful to group L-phrases into CHAINs whose members have PROPER OCCURRENCEs beginning at the same STARTs. Clearly, two L-PHRASEs are so related if one is the *only* right extension of its length of the other in L.

Each CHAIN has a unique shortest member, which is a TWIN, and a unique longest member, which is an END or a BRANCH. Conversely, each TWIN is the shortest member of its CHAIN, and each END or BRANCH is the longest member of its CHAIN. Each CHAIN consists of those L-PHRASEs, and only those, which are left parts of the longest member and right extensions of the shortest member. In the example of Figure 1, the sequence A, AB is a CHAIN. Its shortest member is the TWIN A; its longest member is the BRANCH AB; and the members of the CHAIN have occurrences beginning at the STARTs located in the text at addresses 1 and 4. The TWIN extension ABA of AB is the shortest member of another CHAIN, consisting of ABA, ABAB, and ABABB. The longest member of this CHAIN is the END ABABB. Members of this CHAIN have occurrences in the TEXT beginning only at the START located at text address 4.

The reader will note the similarity in the structure of the index to Fredkin's [1]

Trie Structure. PATRICIA's index differs from Fredkin's Binary Trie structure in that the index records only true branches; where a phrase has only one proper right extension, it is not recorded in the index. This fact reduces the number of index rows to only twice the number of starts, and makes it independent of the length of the stored phrases.

4. *Numbering of Starts, Ends, Branches, Chains, and Twins; Coordinates of L-Phrases*

Since the index to be constructed by PATRICIA will include only numbers and no L-phrases, it is necessary to devise a system of coordinates for L-phrases so that they can be referred to by their coordinates, and so that one can deduce from the coordinates of a phrase, and from data stored in the index, what the phrase is, where its occurrences are in the text, and what the coordinates of its right extensions in L are. Such a system of coordinates is described in this section. Each L-phrase p will have two coordinates: a *chain* coordinate $c(p)$, which it shares with all the members of its chain, and a *length* coordinate $l(p)$, which denotes the length of p. Since no two members of a chain have the same length, the coordinate pair $(c(p), l(p))$ serves uniquely to identify the L-phrase p.

The foundation of the coordinate system is an assignment of *positive odd integers* to starts. This assignment is arbitrary; it can be made by any rule the librarian or the programmer cares to use. Usually it will be made by assigning successive odd numbers to the starts as they are encountered in the text as it is scanned from left to right. This method of assignment is not necessary, however. If the librarian would prefer, as an afterthought, to designate as starts some text addresses not so designated on a first pass through the text and to assign to these, odd numbers larger than those previously assigned to starts further to the right, that is permitted.

In the example in Figure 1, the assignment is as follows:

$$\text{START}(1) = 1$$
$$\text{START}(3) = 4$$
$$\text{START}(5) = 9$$
$$\text{START}(7) = 11$$

This assignment is recorded in the part of the index labeled "START INDEX".

To assign numbers to ends, it is necessary only to note that ends and starts are in one-to-one correspondence, each end corresponding to the unique start at which an occurrence of it begins. (Recall that Rule 1 requires that each end begin at one and only one start.) The odd number assigned to an end is the same as the odd number assigned to the corresponding start. Thus, in the example of Figure 1:

$$\text{END}(1) = \text{ABBABABB}$$
$$\text{END}(3) = \text{ABABB}$$
$$\text{END}(5) = \text{BABABBA}$$
$$\text{END}(7) = \text{BABBA}$$

The odd number assigned to an end e will also serve as the number of the chain and the chain coordinate $c(p)$ of the L-phrases in the chain of which e is the longest member. Thus, for example, $c(\text{ABA}) = c(\text{ABAB}) = c(\text{ABABB}) = 3$, since ABABB = END(3) and ABA, ABAB are all members of the same chain, ABABB being the longest member. We now need a systematic way to assign numbers to branches and thus to the chains of which they are the longest members. To do this we need the following definition.

Definition. The STEM of an end, END(n) (where n is an odd number greater than 1), is the longest left part of END(n), which is also a left part of END(j) for some odd number j, less than n.

In the example,

$$\text{STEM(END(3))} = \text{AB}$$

$$\text{STEM(END(5))} = \Omega$$

$$\text{STEM(END(7))} = \text{BAB}$$

because AB is the longest left part of END(3), which is also a left part of END(1); Ω is the longest left part of END(5), which is also a left part of END(1) or END(3); and BAB is the longest left part of END(7), which is also a left part of END(1), END(3), or END(5).

It can be shown that every branch is the stem of one and only one end and that the stem of every end except END(1) is a branch. It follows that if we define branch numbers, as follows, for each end number n greater than 1,

$$\text{BRANCH}(n - 1) = \text{STEM}(\text{END}(n)),$$

then *even* numbers are assigned in one-to-one fashion to the branches. In the example this rule assigns numbers to branches as follows:

$$\text{BRANCH(2)} = \text{STEM(END(3))} = \text{AB}$$

$$\text{BRANCH(4)} = \text{STEM(END(5))} = \Omega$$

$$\text{BRANCH(6)} = \text{STEM(END(7))} = \text{BAB}$$

The even number assigned to a branch is also assigned to the chain and as a chain coordinate to all the members of the chain of which the branch is the longest member. Thus, for example, $c(\text{A}) = c(\text{AB}) = 2$. Since AB = BRANCH(2), A and AB are in the same chain and AB is the longest member of that chain.

Since every chain has either a branch or an end as its longest member, and since we have assigned numbers to every end and every branch, we have assigned chain numbers to every chain and chain coordinates to every L-phrase. Moreover, because of the manner of assignment, we can deduce several useful relations among chain coordinates of phrases and the start numbers at which occurrences of those phrases in the text begin. Specifically,

(1) If an L-phrase p has an odd chain number, n, then there is one and only one start, namely START(n), at which a proper occurrence of p in the text begins.

(2) If an L-phrase q has an even chain number m, then there are at least two starts at which proper occurrences of q begin. One such start is START($m + 1$). There is at least one other at START(j) for some odd number j, less than m.

Since the phrases in a chain with an even chain number and in the chain with the next larger odd chain number have a common start, the START INDEX lists one start for each such pair of chains. In the example, the phrases in CHAIN(1) have a common start at START(1) = 1; the phrases in CHAIN(2) and CHAIN(3) have a common start at START(3) = 4; the phrases in CHAIN(4) and CHAIN(5) have a common start at START(5) = 9; and the phrases in CHAIN(6) and CHAIN (7) have a common start at START(7) = 11. Note also that the starts listed above are the *only* starts for the phrases with odd chain numbers, but are not the only starts for the phrases with even chain numbers.

Our next requirement is a systematic way of numbering twins. Since each branch has an even branch number and has two twin minimal right extensions, it seems natural to assign twin numbers as follows. For each even branch number n, we denote:

$$\begin{aligned} \text{TWIN}(n) &= (\text{BRANCH}(n))\text{A} \\ \text{TWIN}(n+1) &= (\text{BRANCH}(n))\text{B} \end{aligned}$$

Since this assigns numbers to all twins but Ω and uses all the branch and end numbers but 1, it seems natural, furthermore, to denote:

$$\text{TWIN}(1) = \Omega$$

Since every twin is either the right extension by A or B of a branch, or is Ω, we have now assigned twin numbers to all twins, and the numbers assigned to twins are the same numbers as those assigned to chains. It is not, however, true, in general, that the twin number assigned to a twin is the same as its chain coordinate. Thus we need in the index a table, which we call the TWIN INDEX, which specifies for each twin number the corresponding chain coordinate. In the example of Figure 1, TWINS 1, 2, 3, 4, 5, 6, and 7, respectively, have chain coordinates 4, 3, 1, 2, 6, 5, and 7, respectively. Since each chain includes one and only one twin (its shortest member), and the twin numbers and chain coordinates range over the same set of integers, the TWIN INDEX tabulates a permutation on its set of arguments. This permutation is called TC, the TWIN-TO-CHAIN *coordinate transformation.* In the example of Figure 1, we would say that TC(1) = 4, TC(2) = 3, TC(3) = 1, etc.; in general, TWIN(i) is in CHAIN(TC(i)).

The *height,* HEIGHT(n) of CHAIN(n), is the length of the longest member of CHAIN(n). The third and final component of the index of L is the HEIGHT INDEX, in which is tabulated HEIGHT(n) for each chain coordinate n.

This completes the description of the text and index which describe a library L. In the sections which follow, algorithms are described which operate on the text and index thus far described to retrieve targets characterized by specified keys and to modify the text and index to reflect changes in L.

5. *FINDONE and FINDALL*

In this section two basic algorithms which perform tasks essential to PATRICIA's performance are described. The algorithms and their functions are:

(1) FINDONE, which operates on selected bits of a phrase p and on the twin and height indexes of L to produce a succession of pairs (t, c) which are the twin numbers and chain numbers, respectively, of the twin left parts of p in increasing order of length, *if p is in* L. *In this case,* the last c generated by FINDONE is the chain number of p, and START(c) is a start at which *one*

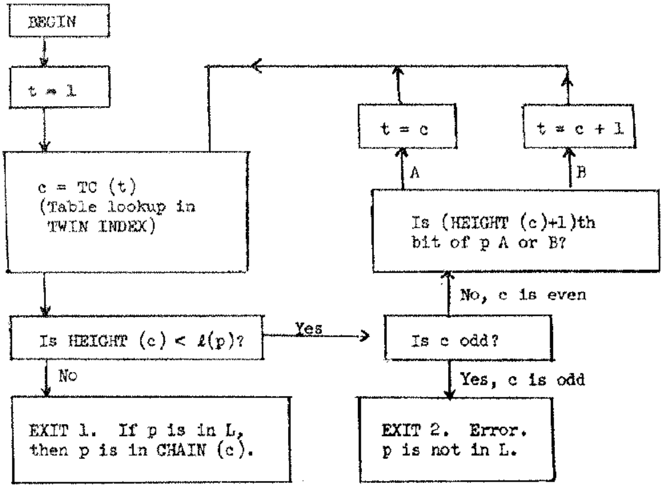


FIG. 2. Flowchart for FINDONE

proper occurrence of p begins. If c is odd, then there are no other proper occurrences of p in L. If c is even, there are others. If p has no proper occurrence in L, then the longest phrase p' in CHAIN(c) shares with p a common left part, q, which is the longest left part of p in L.

(2) FINDALL operates on an even chain coordinate $c(p)$ of a phrase which has more than one proper occurrence in L and on the TWIN INDEX of L to produce a sequence of odd chain coordinates, $c_1, c_2, \cdots, c_n$, which are the chain coordinates of *all* ends which are right extensions of p, in alphabetical order. The starts, START(c_i), are the beginnings of *all* proper occurrences in L of p.

Flowcharts for FINDONE and FINDALL are shown in Figures 2 and 3. Some explanatory notes follow.

FINDONE begins by setting $t = 1$, that is, by noting that TWIN(1) $= \Omega$ is the shortest twin left part of p.

Next, it looks up in the TWIN INDEX the chain coordinate, $c =$ TC(t) of TWIN(t). Assume, for the moment, that p *is* in the library. Later we examine the consequences of FINDONE if p is not in L. Given that TWIN(t) in CHAIN(c) is a left part of p, there are two possibilities: either (1) p is in CHAIN(c), or (2) p is a right extension of the longest phrase, BRANCH(c) in CHAIN(c). FINDONE chooses between these two possibilities by comparing HEIGHT(c) (the length of BRANCH(c) or END(c), the longest phrase in CHAIN(c)) with $l(p)$ (the length of p), and by testing the parity of c. If HEIGHT(c) $\geqq l(p)$, then (1) holds and FINDONE exits at EXIT 1. If HEIGHT(c) $< l(p)$, and c is odd, then FINDONE senses a contradiction, for p is not in CHAIN(c); and the longest phrase in CHAIN(c) is END(c), which has no right extensions in L. The contradiction implies that p is not in L, so FINDONE exists at EXIT 2. If HEIGHT(c) $< l(p)$ and c is even, then FINDONE concludes that (2) holds; p is an extension of BRANCH(c).

The next objective is to decide which of the twin extensions, TWIN(c) and TWIN($c + 1$) of BRANCH(c), is a left part of p. Since they differ only in their last (HEIGHT(c) + 1)-th letter, this decision is made by testing the (HEIGHT(c) + 1)-th bit of p. If it is an A, then TWIN(c) = (BRANCH(c))A is the selected twin;

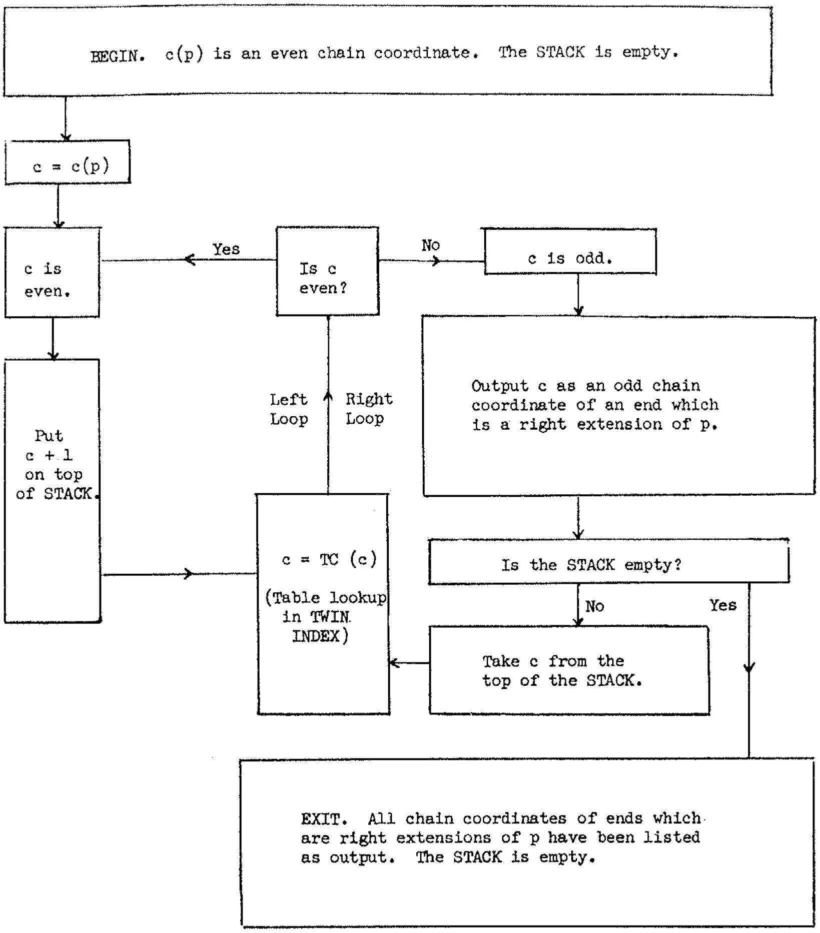

BEGIN. c(p) is an even chain coordinate. The STACK is empty.
c = c(p)
c is even.
Yes
Is c even?
No
c is odd.
Output c as an odd chain coordinate of an end which is a right extension of p.
Left Loop
Right Loop
Put c + 1 on top of STACK.
c = TC (c)
(Table lookup in TWIN INDEX)
Is the STACK empty?
No
Yes
Take c from the top of the STACK.
EXIT. All chain coordinates of ends which are right extensions of p have been listed as output. The STACK is empty.

FIG. 3. Flowchart for FINDALL

if B, then TWIN$(c + 1)$ = (BRANCH(c))B is selected. In either case the TWIN number, c or $c + 1$ of the selected twin, is the new value of t, the twin number of the next longer twin left part of p.

This completes the induction and initiates a new execution of the loop.

Consider, now, what happens if p is *not* in L. Then p has a longest left part, q, which is in L.

Consider two cases; q *is* an end, or q *is not* an end.

If q *is* an end, then FINDONE exits at EXIT 2 on discovering that HEIGHT $(c) = l(q) < l(p)$ and c is odd. In this event q is the longest left part of p which is in L and $c = c(q)$.

If q *is not* an end, then we know that q has two minimal right extensions, qx and qy (where x = A, y = B, or vice versa), and qx is in L and is not a left part of p, while qy is not in L and is a left part of p. In this case, one of the chain coordinates encountered by FINDONE will be $c(q) = c(qx)$. The members of CHAIN(c) which are longer than q and also the members of any other chains whose coordinates are later encountered by FINDONE are all right extensions of qx, hence of q, but are not left parts of p, since qx is not a left part of p. In particular, the longest member,

p', of the last chain whose coordinate is encountered by FINDONE shares with p the left part q, which is the longest left part of p in L.

FINDALL makes use of a STACK. A STACK is a sequence of memory locations into which numbers are placed and from which numbers are taken in such a way that the last number in is the first out. The numbers which FINDALL stacks are the odd twin numbers, corresponding to twins whose last letter is B, which are twin extensions of p, the phrase whose chain coordinate, $c(p)$, is the input to FINDALL.

FINDALL begins with an even chain coordinate $c(p)$. It finds a sequence of chain coordinates c, some even and some odd, all chain numbers of chains whose members are extensions of p. Where c is even, FINDALL forms two twin numbers, c and $c + 1$. It "stacks" $c + 1$, the twin number of (BRANCH(c))B, for later processing and converts c, the twin number of (BRANCH(c))A, to a chain number by use of the TWIN INDEX. Where c is odd, FINDALL outputs c as the chain number of an end right extension of p.

FINDALL has two loops: a left loop in which it processes even chain numbers and a right loop in which it processes odd ones. Each execution of the left loop puts one number into the stack, and each execution of the right loop takes one number out. Since FINDALL begins and ends with an empty stack, the two loops are executed the same number of times in each execution of FINDALL.

Since FINDALL processes each even twin number before its odd twin, it examines the A extension of each branch before the B extension. Thus, it examines the end extensions of p in alphabetical order.

Since FINDALL outputs one odd chain coordinate with each execution of its right loop, the two loops are executed the same number of times, and the odd coordinates produced as output are the START numbers of starts at which proper occurrences of p begin, it follows that each loop of FINDALL is executed as many times as there are proper occurrences of p in L.

Since the table lookups in the TWIN INDEX and the EVEN-ODD tests of c are in the intersection of the two loops, each of them occurs twice as many times, in an execution of FINDALL, as the number of proper occurrences of p in L.

6. *How PATRICIA Detects the Presence of a Phrase and Finds Its Proper Occurrences*

Given a phrase p, PATRICIA tests it for proper occurrence in L as follows:

(1) Execute FINDONE, with p as input. The output is a chain coordinate c, which is the chain coordinate $c(p)$ of p if p is in L.

(2) Look in the text at the address START(c). If an occurrence of p is found there, then p has a proper occurrence in L. If none is found, then p has no proper occurrence in L.

If the outcome of (2) is the conclusion that p has a proper occurrence in L, then PATRICIA finds the location of all proper occurrences of p in L as follows:

(3) If the chain coordinate c found in (1) is odd, then p has only one proper occurrence in L, and that occurrence is at START(c).

(4) If the chain coordinate $c = c(p)$ found in (1) is even, then PATRICIA executes FINDALL, with $c(p)$ as input. The output is a stream of odd chain coordinates, $c_1, c_2, \cdots, c_n$. The proper

occurrences of p begin at START(c_1), START(c_2), $\cdots$, START(c_n) and nowhere else. The ends which extend p are those and only those which begin at the aforementioned starts and have length HEIGHT(c_1), HEIGHT(c_2), $\cdots$.

Note that in all of (1), (2), (3), and (4) PATRICIA makes a minimal number of examinations of the text. Only in (2) does PATRICIA look at the text without knowing in advance that the key p will be found there. It will not be found there only in case p has no proper occurrence in L. In that case, PATRICIA has looked at only one place in the text.

These considerations are of interest where storage space for text is at a premium, and it may be necessary to place the text in a portion of memory not as readily accessible as the main memory—say a disk or magnetic tape. In this situation it is helpful to know that PATRICIA takes only one look at the text to determine whether p is in it or not, and only as many looks as there are proper occurrences of p to recover all proper occurrences and their sequels.

Prior to looking at the text, PATRICIA examines the index in the course of executing FINDONE and FINDALL. Hopefully, the index can be stored in a more readily accessible portion of the memory than the text. Whether it is or not, it is helpful to know how many looks at the index PATRICIA takes to execute FINDONE and FINDALL. In the execution of FINDONE, the chains whose coordinates are examined are chains with monotonely increasing height, and all but the last have height less than the length of p. Thus the loop of FINDONE cannot be executed more times than $l(p) + 1$. In most practical cases, the number of such executions is substantially less than $l(p)$, especially where $l(p)$ is large. This is true because of the redundancy of the language. If the language appearing in L is a reasonably random sample of the possible sequences of A's and B's, then one can expect that an average number of executions of the loop of FINDONE per execution of FINDONE would be asymptotic to $\log_2(l(p))$. This would also be the average number of table lookups in the TWIN INDEX, table lookups in the HEIGHT INDEX, and of each of the other computations which occur in the loop of FINDONE.

Similarly, the number of lookups and other functions performed in FINDALL is strictly determined by the number of proper occurrences of p in L. Each instruction in either loop of FINDALL is executed precisely as many times as there are proper occurrences of p in L, and the two instructions which are in both loops, twice that number. These are the table lookups in the TWIN INDEX and the EVEN-ODD tests of c.

7. *Subroutines ADD p and BUILD INDEX*

Suppose that an index exists for a library L and a set of starts, START(1), START(3), START(5), $\cdot\cdot\cdot$, START(n), for some odd integer n. And suppose it is desired to add to the library a new phrase p, not in L, and to make its occurrence a proper one by denoting the place where it begins as a start. The resulting extended library, consisting of L and p and the left parts of p which are not in L, is called L$'$.

The subroutine ADD p modifies the text and index of L to form a text and index for L$'$ as follows. See Figure 4.

(1) If p is a right part of a phrase already in the text and has been recognized by FINDSTART as a key whose present occurrence should be recognized as a proper occurrence, then proceed

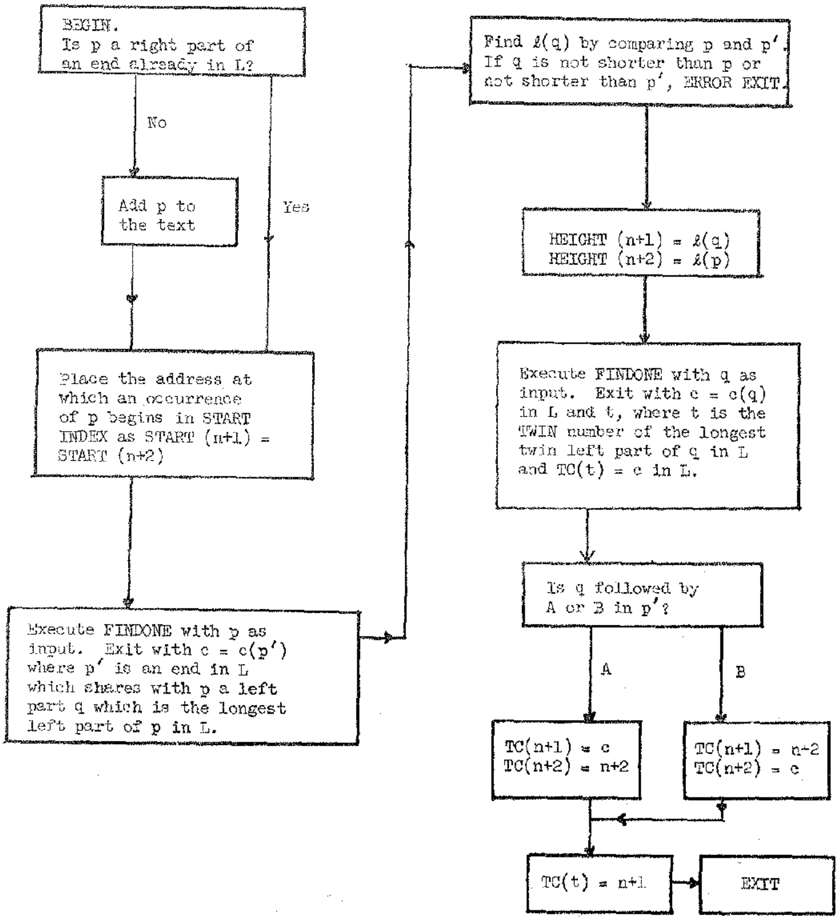

BEGIN.
Is p a right part of an end already in L?
No
Yes
Add p to the text
Place the address at which an occurrence of p begins in START INDEX as START (n+1) = START (n+2)
Execute FINDONE with p as input. Exit with c = c(p′) where p′ is an end in L which shares with p a left part q which is the longest left part of p in L.
Find ℓ(q) by comparing p and p′. If q is not shorter than p or not shorter than p′, ERROR EXIT.
HEIGHT (n+1) = ℓ(q)
HEIGHT (n+2) = ℓ(p)
Execute FINDONE with q as input. Exit with c = c(q) in L and t, where t is the TWIN number of the longest twin left part of q in L and TC(t) = c in L.
Is q followed by A or B in p′?
A
B
TC(n+1) = c
TC(n+2) = n+2
TC(n+1) = n+2
TC(n+2) = c
TC(t) = n+1
EXIT

FIG. 4. Flowchart for ADD p

to (2). If p is not already in the text, then add it at the right of the previously recorded text, or at any other place where there is room for it.

(2) Recognize the newly encountered occurrence of p as a proper occurrence by designating p as END$(n + 2)$ and entering its starting address in the START INDEX as START$(n + 2)$ and START$(n + 1)$.

(3) Execute FINDONE with p as input. This yields a chain coordinate $c = c(p')$ where $p' =$ END(c) is an end which shares with p a longest left part q, which is the longest left part q of p which is in L. Compare p and p' to determine the length of q. Two possibilities arise here, either of which indicates an imminent violation of Rule 1.

(a) p may be already in L. In this case, the comparison of p and p' will reveal that p is a left part of p' and therefore already has a proper occurrence in L. It would, in this case, be a violation of Rule 1 to permit a second proper occurrence of p. (See Sections 8 and 9 for ways to avoid this possibility.)

(b) p' may be an end already in L and a left part of p. This situation is revealed when c is odd and $l(q) =$ HEIGHT(c). In this situation, adding a proper occurrence of p would introduce a second proper occurrence of p's left part, p'. This would also violate Rule 1. (See Sections 8 and 9 for ways to avoid this possibility.)

If neither of the situations described in (a) or (b) arises, then p and p' are both extensions of q; one is an extension of qA and the other of qB, and one and only one of these, qx, is in L. The other, qy, is the shortest left part of p not in L. In this case, go on to (4).

(4) If $l(q)$ < HEIGHT(c) = $l(p')$, and $l(q) < l(p)$, then extend the HEIGHT INDEX by entering:

$$\text{HEIGHT}(n + 1) = l(q)$$

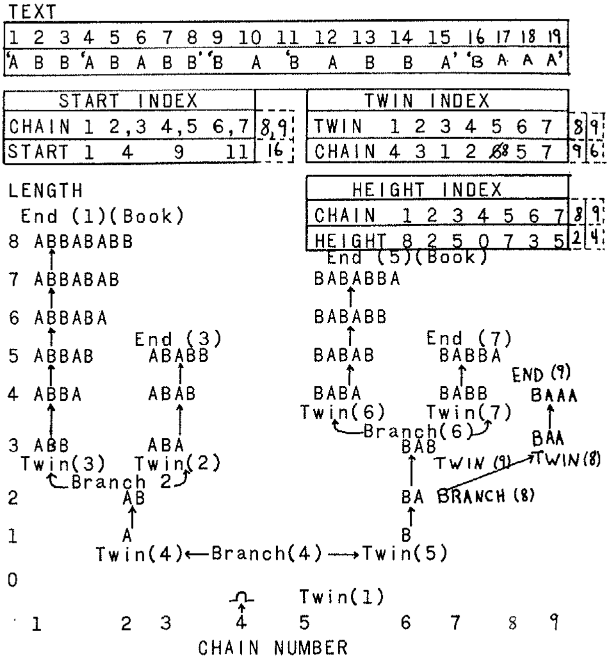
$$\text{HEIGHT}(n + 2) = l(p)$$

This step, together with step (2), identifies p and q, respectively, as the new END$(n + 2)$ and its stem, BRANCH$(n + 1)$.

(5) Execute FINDONE with q as input. Since the presence of q in L is already established in (3), the output of FINDONE consists of a 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). The addition of p to L has made q a branch and has split CHAIN (c) of L into two chains of 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. The members of CHAIN(c) of L which are *shorter* than qx form the new CHAIN $(n + 1)$ of L′, and q is the longest member of that chain. This fact is recognized by replacing the c which occurs in the TWIN INDEX as TC(t) by an $n + 1$.

(6) Extend the twin table by entering c and $n + 2$ as the new chain coordinates of the new twin extensions qA = TWIN$(n + 1)$ and qB = TWIN$(n + 2)$ of the new BRANCH$(n + 1)$ = q. Which of the two acquires which chain coordinate depends on which of the two was already in L and which has just entered L′ as a left part of p. (See (3) above.)

The index of L has now been modified to an index of L′. Figure 5 is a modification of Figure 1, showing the changes in Figure 1 which result from adding a new phrase p = BAAA to the previously existing library and executing ADD p. In this example, q = BA is the new branch, the stem of the new end, BAAA.



TEXT

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
‘A	B	B	‘A	B	A	B	B’	‘B	A	‘B	A	B	B	A’	‘B	A	A	A’

START INDEX					
CHAIN	1	2,3	4,5	6,7	8,9
START	1	4	9	11	16

TWIN INDEX									
TWIN	1	2	3	4	5	6	7	8	9
CHAIN	4	3	1	2	~~6~~8	5	7	9	6

HEIGHT INDEX									
CHAIN	1	2	3	4	5	6	7	8	9
HEIGHT	8	2	5	0	7	3	5	2	4

FIG. 5. Representations of a right library after modification by ADD p

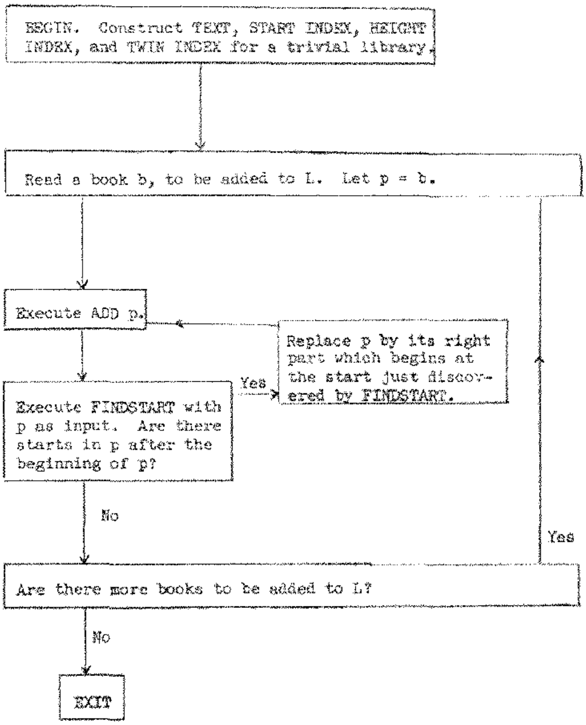

BEGIN. Construct TEXT, START INDEX, HEIGHT INDEX, and TWIN INDEX for a trivial library.
Read a book b, to be added to L. Let p = b.
Execute ADD p.
Replace p by its right part which begins at the start just discovered by FINDSTART.
Yes
Execute FINDSTART with p as input. Are there starts in p after the beginning of p?
No
Yes
Are there more books to be added to L?
No
EXIT

Fig. 6. Flowchart for BUILD INDEX

BUILD INDEX (see Figure 6) begins with an index for a trivial library and extends it by successive applications of ADD p to an index for an arbitrary library L. It reads successive books to be included in L, scanning each of them by FINDSTART, the subroutine specified by the librarian for identifying starts within an input book. It executes ADD p once for each start recognized, including one execution for the start automatically recognized at the beginning of each input book.

8. *Rule 2*

Rule 1 requires that each END shall not have more than one proper occurrence in the library L. This is necessary to insure that an END can be uniquely identified with the START at which its occurrence begins and the BRANCH with the END whose STEM it is. The entire coordinate system upon which PATRICIA operates depends on these relations.

There are several very simple ways to insure compliance with Rule 1. One of them is:

> Rule 2. Each book ends with a sequence of symbols called its *identifier*, which occurs at no other place and within which there are no starts.

If Rule 2 is observed, then every end includes the identifier of the book of which it is a right part and is, therefore, a part of no other book. Since the ends which are right parts of the same book must have different lengths they must all have different starting places, and compliance with Rule 1 is assured.

Rule 2 is probably the simplest way of insuring compliance with Rule 1 and may be the most efficient way. There are many ways in which identifiers can be generated for use in Rule 1. They can be serial numbers generated by BUILD INDEX and attached to each book before it is operated on by ADD p, or they could include START addresses of the book or accession numbers defined by some system in use by the library.

If they include START addresses, then PATRICIA can, by scanning an END(e) of which a key p is a left part, determine where the book b, of which e is a right part, begins, and return as output all of b.

9. *Reverse Library*

In this section the *reverse library technique* is described; this is an alternative to Rule 2 for complying with Rule 1. Its description is much more complex than that of Rule 2, and its use is recommended only in certain situations where compliance with Rule 2 is cumbersome or inefficient. These are situations in which L is very large, so that available memory is a limiting factor, and/or the quantity of computation called for by BUILD INDEX is exorbitant. They are situations in which, furthermore, L includes numerous instances of right parts common to many books whose many occurrences should, in the librarian's judgment, all be retrieved when keys occurring within them are presented. An example of such a situation is a business directory in which many entries terminate with a business address whose last part is common to many entries. Another example is a bibliography in which many entries terminate with a common publisher's name or journal name. While Rule 2 will handle such examples, it does so by adding identifiers to differentiate the many entries with identical endings and by entering into the index the start from each of them. This procedure generates large numbers of starts associated with nearly identical ends and thereby expands the index. The reverse library technique permits retrieval of all such endings, even though only the first occurrence of each end is designated as a proper occurrence. This is achieved at a cost of one additional start per book and the associated additional index data and indexing computation.

It is useful, in applying the reverse library technique, to select a symbol, say $\wedge$, to be called the STOP SYMBOL, which will be a right part of every book, hence of every end, and *which will have no other occurrences*. This insures partial compliance with Rule 1, in that ends will have no occurrences as nonends. Full compliance with Rule 1, however, still requires that each end be restricted to one proper occurrence as an end.

Before going into a detailed description it is helpful to describe, in general terms, how the reverse library technique accomplishes its purpose in a particular example. Consider the library L whose books are:

I SEE THE BIG BLACK BEAR$\wedge$
I SEE THE BIG BROWN BEAR$\wedge$
I SEE THE LITTLE BLACK BEAR$\wedge$
I SEE THE LITTLE BROWN BEAR$\wedge$

Suppose that starts have been identified at the beginning of each word, except at the second occurrences of BLACK and BROWN and at the second, third, and fourth occurrences of BEAR∧, which are prohibited as starts by Rule 1. Confronted with this library, PATRICIA will build an index. On being asked for all right inextendible right extensions of BLACK, PATRICIA will respond that there is exactly one, namely, BLACK BEAR∧. PATRICIA will not point out, however, that BLACK BEAR∧ has occurrences as a right part of more than one book and will furnish a text address for only one of the occurrences.

But now suppose that we augment the library L by including not only the four books listed above, with starts as indicated, but also four more, consisting of the first four *read backward*, with starts only at the beginnings of the reverse books.

∧ЯAƎB ꓘƆA⅃B GIB ƎHT ƎƎƧ I
∧ЯAƎB ꓘƆA⅃B Ǝ⅃TTI⅃ ƎHT ƎƎƧ I
∧ЯAƎB ИWOЯB GIB ƎHT ƎƎƧ I
∧ЯAƎB ИWOЯB Ǝ⅃TTI⅃ ƎHT ƎƎƧ I

Now suppose we wish to find all occurrences of BLACK. PATRICIA first finds all right inextendible right extensions of BLACK. There is only one, BLACK BEAR∧. Turning this around yields ∧ЯAƎB ꓘƆA⅃B. PATRICIA then finds all right inextendible right extensions of ∧ЯAƎB ꓘƆA⅃B. These are:

∧ЯAƎB ꓘƆA⅃B GIB ƎHT ƎƎƧ I
∧ЯAƎB ꓘƆA⅃B Ǝ⅃TTI⅃ ƎHT ƎƎƧ I

Turning this around yields:

I SEE THE BIG BLACK BEAR∧
I SEE THE LITTLE BLACK BEAR∧

which is a complete list of all *left* inextendible *left* extensions of right inextendible right extensions (that is of all *books* which include occurrences) of the word BLACK. Similarly, interrogated with the key, BEAR, PATRICIA will learn that BEAR has only one right inextendible right extension, BEAR∧. It will then reverse this and find that its reverse, ∧ЯAƎB, has four right inextendible right extensions; and their reverses are all the four books in L.

PATRICIA performs this feat without actually storing anything backward or turning anything around; it simply builds a part of its index while reading the text from right to left, and part while reading from left to right. Furthermore, since it is only *right inextendible* L-phrases whose left inextendible left extensions are sought, and since these all terminate at the right ends of books, it is only at the beginnings of the reverse books that starts are needed. Thus the reverse library technique adds only one start per book to the index, regardless of how many starts per book there were in the original library L.

Now let us consider in detail what PATRICIA requires in the form of TEXT, INDEX, and subroutines to implement the reverse library technique. Required are:

(1) A single text, to be read from left to right when reading the forward library L, and to be ·ead from right to left when reading the backward library, L*.

(2) A single index which indexes the combined library, L ∪ L*.

(3) The subroutine FINDALL, which applies alike to the forward library L and the reverse library L*, because it reads no text.

(4) Variants, FINDONE* and ADD* p, of FINDONE and ADD p, which differ only in the manner in which they read the text or the input phrase p. FINDONE* and ADD* p read the input phrase and text from right to left, rather than from left to right.

(5) A variant of BUILD INDEX, which we call MODIFIED BUILD INDEX, which adds starts in L, essentially as BUILD INDEX does, but also in L* when a new book b is added to L and its reverse, b^* to L*. MODIFIED BUILD INDEX is described in detail later; its flowchart is shown in Figure 7.

(6) An index component, called the REVERSE INDEX, which relates chains in L to chains in L*. The REVERSE INDEX is a tabulation of the function REVCH defined for all odd chain coordinates n by

$$\text{REVCH}(n) = c(\text{END}^*(n))$$

where END*(n) is the reverse of END(n), the longest phrase in CHAIN(c). That is, END*(n) is END(n) read backward. END*(n) is in L* if END(n) is in L, and REVCH(n) is its chain coordinate. If END(n) is in L*, then END(n) is a reverse book, END*(n) is the corresponding forward book, and REVCH(n) is the number of the chain in L whose longest member is the forward book END*(n).

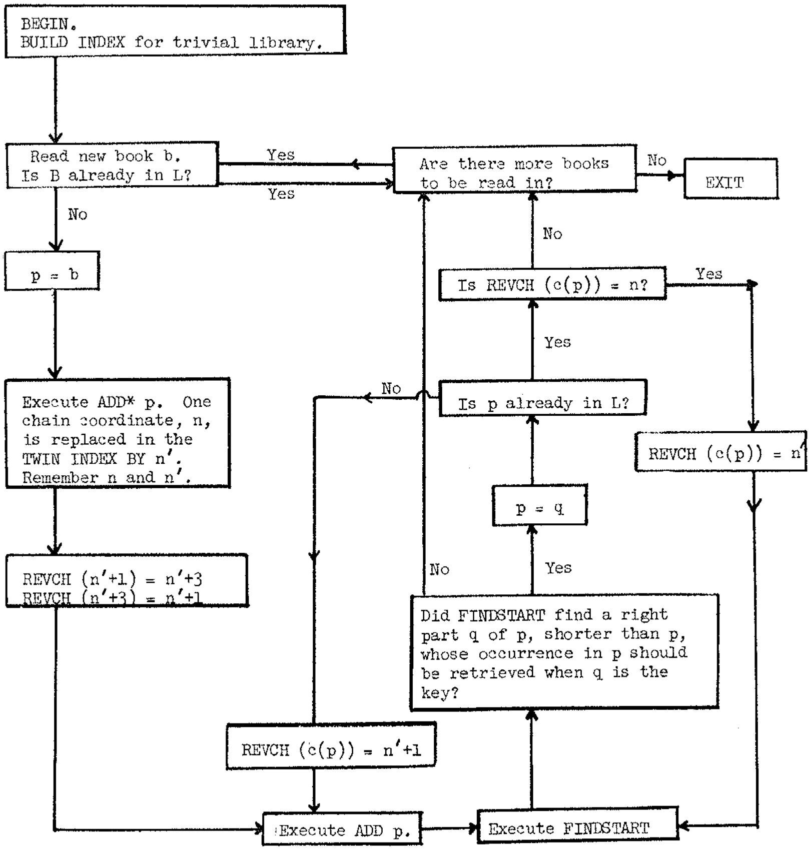

BEGIN.
BUILD INDEX for trivial library.
Read new book b.
Is B already in L?
Yes
Yes
Are there more books to be read in?
No
EXIT
No
p = b
No
Is REVCH (c(p)) = n?
Yes
Yes
Execute ADD* p. One chain coordinate, n, is replaced in the TWIN INDEX BY n'. Remember n and n'.
No
Is p already in L?
REVCH (c(p)) = n'
p = q
REVCH (n'+1) = n'+3
REVCH (n'+3) = n'+1
No
Yes
Did FINDSTART find a right part q of p, shorter than p, whose occurrence in p should be retrieved when q is the key?
REVCH (c(p)) = n'+1
Execute ADD p.
Execute FINDSTART

FIG. 7. Flowchart for modified BUILD INDEX

In the implementation to be described, a book b in L and its reverse, b^* in L*, will have consecutive odd chain numbers; if $b^* = \mathrm{END}(n)$, then $b = \mathrm{END}(n + 2)$.

For any odd n, whether END(n) is a whole book or a right part of a book, the one proper occurrence of END(n) in the text begins at START(n) and ends at START(REVCH(n)), where the reverse of the book it is in begins. Its length, therefore, is

$$\mathrm{HEIGHT}(n) = \pm(\mathrm{START}(\mathrm{REVCH}(n)) - \mathrm{START}(n) + 1),$$

where the sign depends on whether END(n) is in L or L*: plus if in L, minus if in L*. Since, for odd n, HEIGHT(n) can be readily computed from the START and REVERSE indexes, it is not necessary to store HEIGHT(n) for odd n. The memory space assigned to HEIGHT(n) for odd n can, therefore, be used instead to store REVCH(n). Thus the reverse library technique requires no more index space per start than the implementation described earlier.

Remarks on MODIFIED BUILD INDEX follow (see Figure 7). MODIFIED BUILD INDEX operates essentially as BUILD INDEX, except that it adds, through ADD* p, the reverses of each new book to the index, as well as the new book and the right parts of it selected by FINDSTART. MODIFIED BUILD INDEX also builds the REVERSE INDEX by computing the function REVCH. To do this it has to note the branch coordinate, n', of the stem of the most recently added reverse book and the chain coordinate n which that stem had before the book was added. As noted earlier, where b is a book, REVCH transposes the chain coordinates $n' + 3$ and $n' + 1$ of b and its reverse b^*. When p is a right part of the most recently added book b, *and of no earlier book*, then REVCH($c(p)$) is the chain coordinate $n' + 1$ of the most recently added reverse book. When, however, p is a right part of the newest book, *and also of an earlier book*, then Rule 1 prohibits the designation of the new occurrence as a new start. In this case, ADD p is not executed, and the end p is not added again. In this situation, p is already in L, and its chain coordinate has already been assigned a reverse by REVCH. It is possible, however, that this assignment was rendered obsolete by the earlier replacement of n by n' as the chain coordinate of certain phrases which are right parts of the newest book. This possibility is investigated by testing whether REVCH($c(p)$) = n. If it is, then the value n *is* obsolete and is replaced by n'. If the test fails, then the REVERSE INDEX is up to date for p, since no other chain coordinates of ends in L* which were present before the last book was added have been altered. In this case the same remarks apply to all properly occurring right parts of p, and the loop is completed.

Confusion may arise as a result of phrases which are in L ∩ L*, that is, in both L and L*. Confusion may also arise if a phrase p is presented as a key, in search of extensions in L, which has, in fact, extensions in L*. Ways to avoid this kind of confusion are described below.

It is inevitable that there will be phrases which are in both L and L*. One such phrase is Ω. Unless L is a trivial library, there will be at least a few others. Unless L is a very unusual library, which includes many phrases read both forward and backward, there will be at most a few phrases which are in both L and L*.

The stop mark, $\wedge$, and its utility are described above. The stop mark, $\wedge$, has a reverse, $\wedge^*$. Since $\wedge$ is a right part of every book in L, it follows that its reverse $\wedge^*$ is a left part of every book in L* and therefore of every L*-phrase longer than

$\wedge$. If we prohibit proper occurrences of $\wedge^*$ in L, then the phrases which are in L $\cap$ L* will all be left parts of $\wedge^*$, and, presuming that $\wedge^*$ consists of only a few bits, say six, the number of left parts of $\wedge^*$ will be small. None of them will be right inextendible in L $\cup$ L*, so we are guaranteed at least that the ends in L $\cup$ L* are all in L or L* but not in both. It follows that ends in L are not in L*; and their left inextendible left extensions which are retrieved by PATRICIA are all in L, not in L*. There is still a possibility, however, that a key, *p*, will be entered which is in fact a left part of $\wedge^*$ and in L $\cap$ L*. Presented with such a key, unless there are rules or program checks to prevent it, PATRICIA will return the boundaries of all books in L or L* which extend *p* and will look for the books in the text. In those instances in which the book PATRICIA finds is in L*, the beginning boundary will be to the *right* of the end boundary. How the computer will handle this situation depends on the nature of the beast. A recommended preventative is to reject keys which are shorter than $\wedge$ or which include $\wedge^*$ as a left part.

10. *Computer Implementation*

All the algorithms described have been programmed for the CDC 3600 computer and UNIVAC 1108 computers and applied to several substantial information-retrieval problems. Reference [2] is an expanded version of this paper, which includes descriptions of these programs. Reference [3] is an earlier mathematical paper in which the basic concepts used in the design of PATRICIA were formulated.

ACKNOWLEDGMENT. The author acknowledges the valuable assistance of many colleagues, especially that of Mrs. Charlene Lanford, who programmed all the currently running versions of PATRICIA.

REFERENCES

1. FREDKIN, E. Trie memory. *Comm. ACM 3*, 9 (Sept. 1961), 490-500.
2. MORRISON, DONALD R. PATRICIA—*P*ractical *A*lgorithm *t*o *R*etrieve *I*nformation *C*oded *i*n *A*lphanumeric. Res. Rep. SC-RR-67-734, Sandia Corp., Albuquerque, N. Mex., Oct. 1967.
3. MORRISON, DONALD R. A library automaton. Unpublished notes.

RECEIVED NOVEMBER, 1967; REVISED FEBRUARY, 1968