The Complexity of Trie Index Construction

DOUGLAS COMER AND RAVI SETHI

The Pennsylvania State University, University Park, Pennsylvania

ABSTRACT Trie structures are a convenient way of indexing files in which a key consists of a number of attributes Records correspond to leaves in the trie Retrieval proceeds by following a path from the root to a leaf, the choice of edges being determined by attribute values The size of a trie for a file depends on the order in which attributes are tested It is shown that determining minimal size tries is an NP-complete problem for several variants of tries and that, for tries in which leaf chains are deleted, determining the trie for which average access time is minimal is also an NP-complete problem These results hold even for files in which attribute values are chosen from a binary or ternary alphabet

KEY WORDS AND PHRASES information retrieval, trie indexes, trie size, average search time, complexity

CR CATEGORIES 3 74, 4 33, 5 25

1. *Introduction*

Let a *record* be information of an unspecified nature to be stored and retrieved, and let a *file* be a collection of records. Associated with a record is a *key* that uniquely identifies the record

For the moment, suppose that a key is an integer like 72 or 35. Elegant methods are available for organizing such keys so that questions like "Is this record in the file?" can be answered efficiently. These methods also permit records to be added to or deleted from the file dynamically. Bayer and McCreight [2] have a particularly interesting data structure, called B-trees, for coping with updates. B-trees, or the related 2-3 trees of Hopcroft reported in Aho et al. [1], permit a record to be retrieved, added, or deleted from an n-record file in $O(\log n)$ time. Knuth [10] is a general reference for indexing with integer keys.

Key comparison based methods like B-tree searches have received wide attention in the literature and are relatively well understood. This happy situation does not apply to the case where a key consists of a number of attributes, as is true when a key is given by a k-tuple $(i_1, i_2, \dots, i_k)$. B-tree searches and related methods rely on comparing whole keys in constant time. When a key is given by a k-tuple, comparison of keys is no longer an elementary operation in standard measures of complexity. A common response is therefore to determine some subset of the attributes on which to index. Schkolnick [12] considers a probabilistic model for selecting an appropriate subset.

Another approach for handling multiattribute keys is to use "trie" based methods proposed by de la Briandais [5] and Fredkin [7]. Consider the set of strings

back	bane	bank	bare	barn
band	bang	barb	bark	been

The first letter in each string is b. Examination of the second letter splits the set into two subsets: the strings starting with ba and the string starting with be. The process of

Authors' present addresses D Comer, Computer Sciences Department, Purdue University, West Lafayette, IN 47907, R Sethi, Bell Laboratories, 600 Mountain Ave , Murray Hill, NJ 07974

Journal of the Association for Computing Machinery, Vol 24, No 3, July 1977, pp 428-440

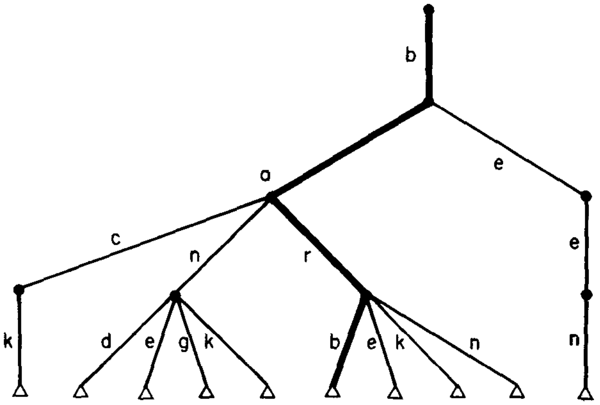


FIG 1 Full trie for the set of strings {back, band, , been} The darkened path gives the sequence of decisions for the string "barb "

splitting is well illustrated by the tree in Figure 1. Fredkin [7] used the name trie (pronounced "try") for these tree structures.

In this paper we are interested in the properties of tries that have a bearing on the size of the trie, or the average access time for a record But first we review some of the issues involved in implementing tries.

A straightforward implementation of the trie in Figure 1 is to represent each nonleaf node u by a 26-element array, or table, with an entry for each letter of the alphabet. The entry for the letter a at node u points to the table for the appropriate son of u This tabular implementation allows the test at node u to be made in constant time. The tables, however, may be quite wasteful in space

In order to save space, de la Briandais [5] proposed a binary tree representation for forests quite similar to that given in Knuth [9]. The idea is to place all sons of a node u in a linked list Node u points to the first element in the list Note that the time spent in selecting the appropriate son of u is no longer constant; at worst, all sons of u may have to be examined.

The linked list implementation was referred to as a "doubly chained tree" by Sussenguth [14], and the term has survived in the literature. Severence [13] and Yao [15] are concerned about the space-time trade-offs of the two representations and consider heuristics for a compromise in which the first few levels are represented by tables and the remaining levels by doubly chained trees.

The tradeoffs between the doubly chained and tabular implementation are strongly influenced by the size of the alphabet used in constructing keys. Most of the results in this paper apply even when the binary alphabet is used With a binary alphabet the doubly chained tree takes up at least as much space as the tabular implementation and might even take more time. Recall that, in a doubly chained tree, going down a sequence of right branches may take twice as long as going down a sequence of left branches

Results in this paper that are concerned with the size of tries apply to both implementations The results concerned with the average access time of records apply to the tabular implementation only. At the same time we wish to emphasize that the average access time results hold even when the binary alphabet is used.

Having reviewed the representation of tries, we turn to the issues addressed in this paper. The study of tries originated with de la Briandais [5] and Fredkin [7], who used alphabetic keys. The order of testing of the letters of a key was understandably left-to-right, as in Figure 1. When we view a key as a k-tuple in which the attributes are unrelated, the left-to-right ordering is no longer as natural. In fact de Maine and Rotwitt [6] note that the order in which attributes are tested may influence the size of the resultant trie. Consider, for example, the trie in Figure 2 constructed from the same

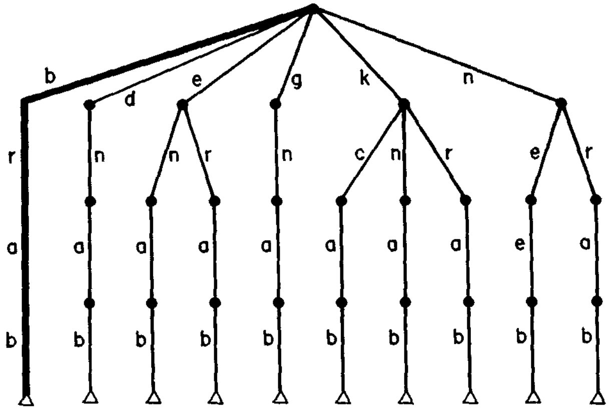


FIG 2 Full trie for the strings in Figure 1 formed by testing the characters from right to left

strings as in Figure 1, but testing letters from right to left The number of nonleaf nodes increases from 8 in Figure 1 to 27 in Figure 2. Thus the trie in Figure 2 clearly takes up more space.

Question What order of testing attributes leads to the smallest trie?

We show that the above question is difficult to answer in a precise sense: Specifically we show that that question represents an NP-complete problem.[1]

Note from Figure 2 that a key with b as its last letter is either barb or it is not in the file Similarly a key with g as its last letter is either bang or it is not in the file. Thus instead of testing all the attributes in a key, we can stop testing the moment we have narrowed down the set of possibilities to one, as in Figure 3(b). One way of viewing the transformation of Figure 2 to 3(b) is to note that all chains that lead to leaves have been pruned. The tries in Figure 3 will be referred to as *pruned tries*.

Question. What order of testing attributes leads to the smallest pruned trie?

We can also observe that subtrees of a trie may contain internal chains as well as chains leading to leaves. By including in each node extra information telling how many attributes to "skip" when searching, these internal chains may be eliminated. Tries from which both internal and leaf chains are removed will be referred to as *collapsed tries.*

Question. What order of testing attributes leads to the smallest collapsed trie?

In a pruned trie the sooner a node is distinguished the faster the search proceeds. In the present example not only does the pruned trie in Figure 3(b) have fewer nonleaf nodes than the trie in Figure 3(a), it calls for faster average search time.

Question What order of testing attributes leads to the least average search time in a pruned trie?

All four of the above problems will be shown to be NP-complete.

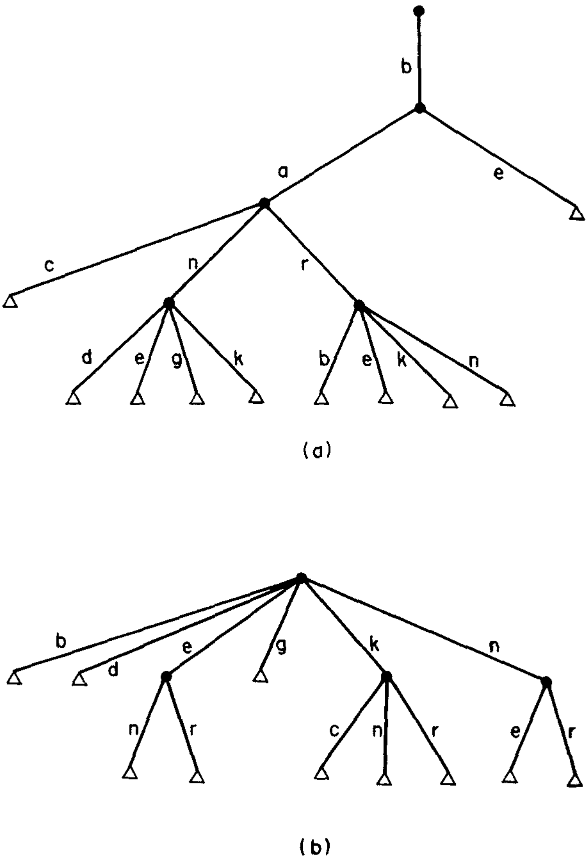
In all the above questions we have concerned ourselves with a global ordering of attribute selection which applies to all paths from the root to a leaf in the trie. We also consider tries in which attributes are tested in different orders along different paths from the root to a leaf. This class of tries will be called *order-containing* or *O-tries* since the order of attribute testing must be contained in the trie itself

Question. Which pruned O-trie is the smallest?

The problem of finding a smallest pruned O-trie will be shown to be NP-complete as a corollary to the proof of pruned trie minimization.

This paper represents a modest beginning in the study of the properties of tries. So far we have only shown that, in a precise sense, the questions above are difficult to answer. Eventually the question of updates must also be confronted

[1] The uninitiated reader would not go too far wrong in viewing the term NP-complete as jargon for "provably difficult" problem Aho et al [1] is a reasonable reference for the subject In Section 2 we discuss the process of showing that a given problem is NP-complete



(a)

(b)

FIG 3 (a) and (b) give the pruned tries for the full tries in Figures 1 and 2, respectively

2. *A Model for Retrieval*

In this section we define the notions of file, key, query, and trie We also identify the problems addressed in this paper and work out a sample proof of NP-completeness.

A file will be thought of as a two-dimensional table with a row for each record and a column for each attribute. Presumably each row contains a pointer to its record, which can be obtained easily once the row has been identified. A query will either be a row in the table or will not be present in the file.

In the examples in Section 1 the attributes were lower case letters; so each attribute could take on 26 possible values We are interested in results that hold even when the number of values an attribute may take on is quite small, such as 2; so we introduce the notion of alphabet size for a file.

Definition. Let $A_1, A_2, \ldots, A_k$ be a finite set of *attributes,* where attribute A_i takes on values from the finite set V_i, $1 \leq i \leq k$. A *file* F is a subset of $V_1 \times V_2 \times \cdots \times V_k$, and a *key* is an element of F. The *alphabet size* of F is given by $\max\{|V_1|, \ldots, |V_k|\}$, where $|V|$ represents the number of elements in V. Files with alphabet sizes 2 and 3 will be referred to as *binary* and *ternary* files, respectively.

A *query* q is an element of $V_1 \times V_2 \times \cdots \times V_k$. □

Graph definitions used throughout this paper are standard; the reader is referred to Aho et al. [1] The next definition defines tries of the kind illustrated in Figures 1 and 2.

Definition. A *full trie* for a file F is a tree with all leaves at depth[2] k such that the following are true.

[2] The root of a tree is at *depth* 0 The sons of a node at depth $i - 1$ are said to be at depth i.

1. Let $A_1, A_2, \ldots, A_k$ be the attributes of F and let π be a permutation of $1, 2, \ldots, k$. All edges leaving a node at depth $i - 1$ have distinct labels chosen from $V_{\pi(i)}$ for all i, $1 \leq i \leq k$.

2. The labels encountered on each path from the root to a leaf correspond to an element of F, and, for each element of F, there is such a path. □

Note that all we need to do to specify a full trie is to specify π, which gives the order in which attributes are tested In addition to full tries, we are interested in tries in which leaf chains have been pruned, as in Figure 3.

Definition. A node u in a tree is the *head* of a *leaf chain* if (a) the father of u has more than one son, and (b) u and all its descendants have at most one son.

A pruned trie for a file F is formed from a full trie T for F by deleting the proper descendents of all nodes u in T such that u is the head of a leaf chain. □

The next definition provides terms that will be convenient in the sequel.

Definition. Let u be the head of a leaf chain in a full trie T. Let p be the path from the root of T to some leaf such that p passes through u. Then the record denoted by the labels on p is said to be *distinguished* at u

Since a full or pruned trie T distinguishes all records in a file F, we say F is *indexed* by T. □

Definition. The *space* taken by a trie T (full or pruned) is the number of nonleaf nodes in T □

In this paper we are interested in the space and search time requirements of tries. Specifically, we consider the following problems.

Problem 1 (Least space full trie).

Given: File F with alphabet size s and an integer j.

Question: Does there exist a full trie for F with space no more than j? □

Problem 2 (Least space pruned trie)

Given: File F with alphabet size s and an integer j.

Question: Does there exist a pruned trie for F with space no more than j? □

We are also interested in the average time taken to access a record. We must be careful in talking about the access time since the time taken to traverse a path depends on the alphabet size and the underlying data structures As mentioned in Section 1, our access time results apply to the table implementation and do not apply to the doubly chained implementation. Nevertheless, our results are strong in that they hold for any alphabet size $s \geq 2$.

Definition. Let T be a pruned trie for a file F. Then the *access time of a leaf* in T is given by the depth of the leaf. The *access time* (*search time*) of T is the sum of the access times of all leaves. □

Note that we can find the average access time for a leaf in T by dividing the access time of T by the number of leaves. Since the number of leaves in a trie is fixed, the division is not necessary for our purposes.

Problem 3 (Least access time pruned trie)

Given: File F with alphabet size s and an integer j.

Question: Does there exist a pruned trie for F with access time no more than j? □

Problems 1 and 3 are NP-complete for all alphabet sizes $s \geq 2$, and problem 2 is NP-complete for all alphabet sizes $s \geq 3$. The proofs are given in Appendixes A, B, and C.

At this point we introduce problem SAT3, which is crucial to demonstrating that problems 1–3 are NP-complete.

Let n be a positive integer and $G_n = \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$. The elements of G_n are called *literals*. Informally a literal in G_n can either be true or false. In defining SAT3 we define clauses c_j, like x_1 *or* x_2 *or* $\bar{x}_3$. A clause is true if one of the literals in it is true.

We refer to the pair $(x_i, \bar{x}_i)$ as a *variable*. The *complement* of x_i is $\bar{x}_i$ and the *complement* of $\bar{x}_i$ is x_i. If a literal y is true, then the complement of y is false and vice versa. Given a set of clauses $c_1, c_2, \ldots, c_m$, the clauses are *satisfiable* if, under some truth assignment to literals in G_n, all clauses are true. In the definition of SAT3 a set H will specify exactly which literals in G_n are true. In order for the truth assignment H to satisfy $c_1, \dots c_m$, for each c_j, one of the literals in H must also be in c_j, i.e $H \cap c_j \neq \varnothing$.

Problem SAT3 (Satisfiability with three literals per clause).

Given: $I = \langle n, c_1, \dots, c_m \rangle$, where n and m are positive integers, $n \leq 3m$, $c_j \subseteq G_n$, and $|c_j| = 3$, for $j = 1, 2, \dots, m$.

Question: Does there exist a set $H = \{y_1, \dots, y_n\}$ such that y_i equals x_i or $\bar{x}_i$ for $1 \leq i \leq n$, and $H \cap c_j \neq \varnothing$ for $j = 1, \dots, m$? If there is such a set for a given I, we say I is satisfiable □

In the above definition we have used $x_1, \bar{x}_1$, .. for literals. Since we are dealing with small examples, we avoid subscripts and use x, $\bar{x}$, y, $\bar{y}$, ... for literals. Moreover, we use "+" to connect literals in a clause and "·" to concatenate clauses. Thus $I = (x + \bar{y} + z)\cdot(\bar{x} + y + z)$ is an instance of SAT3.

A solution to SAT3 would be an algorithm that takes an instance I of SAT3 and answers true if and only if I is satisfiable. Cook [4] showed that SAT3 is NP-complete, or informally SAT3 is known to be a hard problem.

At this stage we shall give the reader a feeling for the constructions used for our results by showing that the least access time problem for pruned tries is at least as difficult as SAT3. For simplicity, we use an alphabet size of 9. In Appendix C the result will be improved so that it holds for binary files also

The file $F(I)$ in Figure 4 has been constructed from the instance $I = (x + \bar{y} + z)\cdot(\bar{x} + y + z)$ of SAT3. I has $n = 3$ variables and $m = 2$ clauses; $F(I)$ has $6n + 4m = 26$ records and $2n + m = 8$ attributes. The subfiles ⟨J:Q⟩,[3] ⟨J:P⟩, and ⟨K:P⟩ depend only on n and m. The subfile ⟨K:Q⟩ is constructed from the clauses $c_1 = (x + \bar{y} + z)$ and $c_2 = (\bar{x} + y + z)$.

The first four records in K, 19–22, are for c_1. For these records, attributes X, $\bar{Y}$, and Z have each been set to 8, 8, 9, and 9. Similarly, in the next four records, for $c_2 = (\bar{x} + y + z)$, attributes $\bar{X}$, Y, and Z have each been set to 8, 8, 9, and 9.

We now show that $F(I)$ has a pruned trie with access time $3n(n + 1) + 2m(m + 1) + 4nm$ if and only if I is satisfiable. The above expression for the access time represents a pruned trie with six leaves at each of the first n depths and four leaves at each of the next m, as in Figure 5.

Suppose that I is satisfiable. In our example $H = \{x, y, \bar{z}\}$ satisfies both $(x + \bar{y} + z)$ and $(\bar{x} + y + z)$. We construct a trie T for $F(I)$ by first testing on the attributes corresponding to elements of H, attributes X, Y, and $\bar{Z}$ in Q. Then we test on each attribute in P. The resultant trie is given in Figure 5. The reader is urged to examine the pruned trie in Figure 5 closely, paying attention to the fact that the four records for each clause are separated into two groups by the attributes corresponding to elements in H.

Now suppose that I is not satisfiable. We claim that at least $n + 1$ attributes must be chosen from the set Q. The reasoning is as follows In order to distinguish the first $6n$ records, at least one attribute must be chosen for each pair of complementary literals If exactly n attributes are chosen, let H' be the set of literals corresponding to these attributes.

Since I is not satisfiable, for some clause c, $H' \cap c = \varnothing$. But then the three columns in Q in which the four records for c have 8, 8, 9, and 9 are not tested So the four records for c are together in a block Testing the attribute for c in the set P cannot distinguish these records We must therefore test on at least one more attribute from Q.

It follows from the above discussion that any pruned trie T for $F(I)$ has depth at least $n + m + 1$. From the construction of $F(I)$, at most $6i$ records can have been distinguished by depth i, $i \leq i \leq n$; at most $6n + 4i$ by depth $n + i$, $1 \leq i \leq m$. Since there is at least one record at depth $n + m + 1$, trie T must have access time at least $3n(n + 1) + 2m(m + 1) + 4nm + 1$.

Therefore, $F(I)$ has a pruned trie with access time $3n(n + 1) + 2m(m + 1) + 4nm$ if and only if I is satisfiable We can now state the following theorem.

[3] ⟨J.Q⟩ refers to the part of the tables delineated by the records in J and the attributes in Q

		Q						P	
		X	$\bar{X}$	Y	$\bar{Y}$	Z	$\bar{Z}$	C_1	C_2
J	1	2	2						
	2	3	3						
	3	4	4						
	4	5	5						
	5	6	6						
	6	7	7						
	7			2	2				
	8			3	3				
	9			4	4				
	10			5	5				
	11			6	6				
	12			7	7				
	13					2	2		
	14					3	3		
	15					4	4		
	16					5	5		
	17					6	6		
	18					7	7		
K	19	8			8	8		8	
	20	8			8	8		9	
	21	9			9	9		8	
	22	9			9	9		9	
	23		8	8		8			8
	24		8	8		8			9
	25		9	9		9			8
	26		9	9		9			9
		x	$\bar{x}$	y	$\bar{y}$	z	$\bar{z}$		

FIG 4 (Least access time pruned trie) The file $F(I)$, where $I = (x + \bar{y} + z)\ (\bar{x} + y + z)$ The attribute corresponding to a given literal x is denoted by the relevant upper-case letter X All unspecified entries are 1

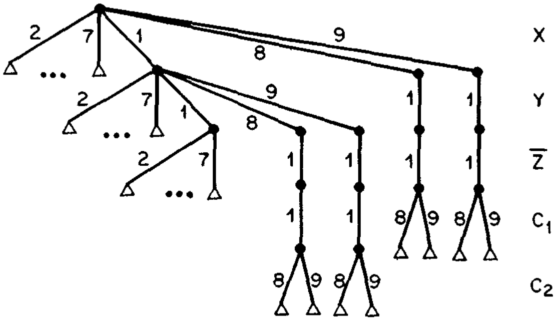


FIG 5 Pruned trie for the file $F(I)$ in Figure 4 Attributes are tested in order X, Y, $\bar{Z}$, C_1, and C_2

THEOREM 1. *The least access time pruned trie problem for files with alphabet size at least 9 is NP-complete.*

PROOF In the discussion before the statement of the theorem we established that the problem on hand was at least as difficult as SAT3. The remaining details of the proof are technicalities and are left to the initiated reader. □

Since all the results in this paper are proved in very much the same manner as the above theorem, we have relegated them to the Appendixes.

3. *Conclusions*

In this paper we have considered two kinds of tries: (a) tries in which each attribute is tested, and (b) tries in which testing of attributes stops when a record has been distinguished. We have demonstrated that determining tries that are minimal in terms of storage space is NP-complete for all alphabet sizes s, $s \geq 2$ for full tries, and $s \geq 3$ for pruned tries, collapsed tries, and pruned O-tries With a tabular implementation for tries, determining a minimal average access time trie is an NP-complete problem for all alphabet sizes $s \geq 2$.

We do wish to point out that the files used to prove the NP-completeness results in this paper are atypical. In the reductions the number of attributes k is about the same as the number of records r In practice we expect k to be much smaller than r In studying heuristics it is therefore desirable to derive performance bounds with both r and k as parameters.

Tries are a natural structure to consider while implementing a retrieval system. The results in this paper suggest that many related trie construction problems are "hard" in a computational sense. We have concentrated on the initial construction of a trie for a file and have not considered addition or deletion of records; further work on trie-based structures that can easily cope with updates is warranted.

Appendix A. Full Tries

Recall that the space taken by a full trie is given by the number of nonleaf nodes in the trie. If T is a full trie for a k-attribute file with r records, then all leaves of T will be at depth k. The space taken by T is given by $\sum_{i=0}^{k-1} b_i$, where b_i is the number of nodes at depth i, $0 \leq i \leq k$. Note that the b_k leaves at depth k are not counted in the space.

Since T is a tree, $b_i \geq b_{i-1}$, the rate at which the b_i sequence grows determines the space taken by T. We define $d_i = b_i - b_{i-1}$, $1 \leq i \leq k$, and $d_0 = b_0 = 1$. Each item in the d_i sequence represents the number of *new* nodes which appear at depth i; the sequence $d_0, d_1, \ldots, d_k$ will be called a *profile* of the trie.[4]

Let us now relate SAT3 to the problem of least space full tries. The file $F(I)$ constructed from an instance of SAT3, $I = (x + \bar{y} + z)\cdot(\bar{x} + y + z)$, is given in Figure 6. $F(I)$ has $2n + 3m$ attributes and $3n + 7m + 1$ records

Before proceeding with the proof of NP-completeness, we examine a useful property of $F(I)$.

LEMMA 1. *Let T be a least space full trie for $F(I)$. Then the profile for T has no 0 elements.*

PROOF. Suppose the profile for T does have 0 elements. We show that T cannot be a least space full trie for $F(I)$

Note from Figure 6 that because of the records in group K, selecting an attribute from set Q will cause at least one new node to appear. Thus a 0 element in the profile must result from the choice of an attribute, say A, in N Each attribute in N has 1 entries for exactly two records. Let the two records for which A has 1 entries be γ_1 and γ_2.

Since selecting attribute A causes no new nodes to appear, one of the following two cases must occur: Either (1) γ_1 and γ_2 are in a block consisting of γ_1 and γ_2, or (2) both γ_1 and γ_2 have been distinguished and are in single record blocks.

Case 1. If γ_1 and γ_2 are together in a block, let B be the attribute with 1 entries for γ_1 and γ_2 chosen before A. Since B has 1 entries for other records, it will not only separate out γ_1 and γ_2 but will also cause at least two new nodes to appear Note that a smaller trie can be formed by interchanging the order in which B and A are chosen. The interchange will result in the profile having two consecutive 1's rather than 2, 0 Thus the new trie takes less space, contradicting the minimality of T.

Case 2. γ_1 and γ_2 have both been distinguished. Let $\bar{A}$ be the attribute in N corresponding to the complement of the literal to which A corresponds. Without loss of generality, let $\bar{A}$ have a 1 entry for γ_1.

Consider the attributes that must be selected for γ_2 (the record for which $\bar{A}$ has a 0 entry) to be distinguished. Aside from A, all other attributes with a 1 entry for γ_2 are in the set Q. Since all attributes in Q have 1 entries for records in K, at least two of them

[4] The connection between the profile and size of a trie is considered in more detail in Comer and Sethi [3], where more detailed proofs of the NP-completeness results are given

must be selected in order to distinguish γ_2. But then, as in case 1, the second such choice adds at least two new nodes. Interchanging the second of these two attributes with A results in a trie taking less space. This is a contradiction.

The lemma must therefore hold. □

The next lemma will aid us in proving a lower bound on the space taken by any trie for $F(I)$.

LEMMA 2 *Let A be an attribute from the set N such that A has* 1 *entries for records γ_1 and γ_2 Then there exists a least space full trie for $F(I)$ in which A is selected before any other attribute with* 1 *entries for γ_1 and γ_2.*

PROOF. From Lemma 1 we can assume that any least space trie T for $F(I)$ has no 0 elements in its profile. Moreover, from the proof of Lemma 1 it follows that there may be at most one attribute B with 1 entries for γ_1 and γ_2 selected before A. Construct a trie T' from T by interchanging the order in which B and A are selected. We claim that T' takes no more space than T.

Let L be the set of seven records from K corresponding to the clause represented by B. From Figure 6 there are two attributes B_2 and B_3 that also have 1 entries for some elements of L. The point to note is that B_2 and B_3 may be selected between B and A in T Since B, B_2, and B_3 are symmetric in their effect on records in L, delaying the selection of B cannot increase the number of nonleaf nodes caused by elements of L.

Since both A and B have 1 entries for γ_1 and γ_2, interchanging the order of A and B does not increase the number of nonleaf nodes. The lemma therefore holds. □

Let us now turn to the relation between I and $F(I)$. In Figure 6 $H = \{x, y, \bar{z}\}$ satisfies I. Selecting attributes X, Y, $\bar{Z}$, C_{11}, C_{22}, $\bar{X}$, $\bar{Y}$, Z, C_{12}, C_{21}, C_{13}, and C_{23} forms a trie for $F(I)$. The first three choices correspond to elements of H; the next two correspond to a literal from each clause that is in H. The remaining attributes from N are selected next, followed by the attributes from Q (which are selected in pairs). This sequence of selections is used below to show that the problem is NP-complete.

THEOREM 2. *The least space full trie problem for binary files is NP-complete.*

PROOF. We reduce SAT3 to the problem on hand. Let I be an instance of SAT3. Construct the file $F(I)$ as shown in Figure 6. Let S denote the space taken by a trie formed as in the example above. We show that $F(I)$ has a trie taking space S if and only if I is satisfiable.

If I is satisfiable, then a trie of size S exists So suppose there is a trie T for $F(I)$ taking space S From Lemma 1, the profile of T has no 0 elements. From Lemma 2 we can assume that an attribute A in N is chosen before any attribute with 1 entries for the 1 entries of A is chosen from Q. Thus the first choice for each pair of attributes in N will lead to one new node and the second to two new nodes. From Figure 6, each triple of attributes in Q will lead to at least one, two, or four new nodes Thus the best possible trie for $F(I)$ has space S. Since there are $n + m$ selections that lead to one new node, n of these pick one of each pair of attributes in N. Let $H \subseteq G_n$ correspond to these attributes in N. The fact that there are at least m more selections with one new node verifies that H satisfies I.

Thus $F(I)$ has a trie taking space S if and only if I is satisfiable. □

Appendix B. Pruned Tries, Collapsed Tries, and Pruned O-Tries

In this Appendix we show that the problem of finding a minimum size pruned trie is NP-complete.

THEOREM 3. *The following problems are NP-complete for all alphabet sizes s, $s \geq 3$: (a) pruned trie space minimization, (b) collapsed trie space minimization, and (c) pruned trie space minimization for O-tries.*

PROOF. The same construction suffices for all three problems. We give the proof for the pruned trie space minimization problem. From the construction in Figure 7, there is an ordering of attributes which yields a trie of $2n + m$ nodes when the formula is

		N						Q					
		x	$\bar{x}$	y	$\bar{y}$	z	$\bar{z}$	c_{11}	c_{12}	c_{13}	c_{21}	c_{22}	c_{23}
J	1	1	1					1			1		
	2	1	0					1			0		
	3	0	1					0			1		
	4			1	1				1			1	
				1	0				0			1	
				0	1				1			0	
						1	1			1			1
						1	0			1			1
	9					0	1			0			0
K	10							1	1	1			
								1	1	0			
								1	0	1			
								1	0	0			
								0	1	1			
								0	1	0			
	16							0	0	1			
	17										1	1	1
											1	1	0
											1	0	1
											1	0	0
											0	1	1
											0	1	0
	23										0	0	1
	24												
								x	$\bar{y}$	z	$\bar{x}$	y	z

FIG 6 (Least space full trie) The file $F(I)$ for $I = (x + \bar{y} + z)\ (\bar{x} + y + z)$ All entries for record 24 are 0 Choosing attributes from set N which correspond to literals which satisfy I will enable m selections from set Q which add only one new node each and a minimum size trie will result

satisfiable. We show that if a trie of $2n + m$ nodes exists, the formulas must be satisfiable.

In order for us to distinguish records in L, T must have depth $2n + m$ and must therefore have at most one nonleaf node at each depth Consideration of the records in L shows that they must be distinguished in order Since exactly one of U_i and $\bar{U}_i$, $1 \le i \le n$, in N must be chosen first, let H be the set of literals corresponding to the attributes chosen in N. Since each record in K is distinguished in T, it must be true that H satisfies I. □

Appendix C. Access Time for Pruned Tries[5]

In Section 2 we showed that the least access time problem for pruned tries is NP-complete for alphabet size 9 Here we tighten the result to hold for all alphabet sizes s, $s \geq 2$. The basic idea is to construct a file for which the least access time pruned trie T is like a full binary tree for the first $h + 1$ depths, where $h = 4 + 2\log(n + m)$. At depth h we have $3n + m$ nonleaf nodes corresponding to groups of records J_{11}, .. , K_m, where the J and K groups are as in Figure 8. There are also $2^h - (3n + m)$ leaves at this depth. (The integer h has been chosen large enough that if the records represented by these leaves are not distinguished by depth $h + 1$, then the trie in question has an access time that is too large) If the first h depths are as mentioned above, then we can simulate the reduction for alphabet size 9 by using a binary alphabet.

The relation between an instance I of SAT3 and a file $F(I)$ is given in Figure 8 We now show that attributes in N must be tested before attributes in Q or P or else the access time will be too large

[5] In a recent paper Hyafil and Rivest [8] show that the problems of leaf access time for collapsed and pruned O-tries is NP-complete for all alphabet sizes s, $s > 2$

	N						Q									P		
	U_1	$\bar{U}_1$	U_2	$\bar{U}_2$	U_3	$\bar{U}_3$	C_{11}	C_{12}	C_{13}	C_{21}	C_{22}	C_{23}	C_{31}	C_{32}	C_{33}	W_1	W_2	W_3
J	1						1						1			1		
		1								1						1		
			1					1			1						1	
				1										1			1	
					1				1			1						1
						1									1			1
K							1	1	1									
										1	1	1						
													1	1	1			
L	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
					2	2	2	2	2	2	2	2	2	2	2	2	2	2
							2	2	2	2	2	2	2	2	2	2	2	2
										2	2	2	2	2	2	2	2	2
													2	2	2	2	2	2
																2	2	2
																	2	2
																		2
	x	$\bar{x}$	y	$\bar{y}$	z	$\bar{z}$	x	y	z	$\bar{x}$	y	$\bar{z}$	x	$\bar{y}$	$\bar{z}$			

FIG 7 Sample construction for least space pruned tries $F(I)$ is shown for $I = (x + y + z)\,(\bar{x} + y + \bar{z})\cdot(x + \bar{y} + \bar{z})$ Lower-case letters give the literal corresponding to an attribute The ordering U_1, $\bar{U}_2$, $\bar{U}_3$, C_{11}, C_{23}, C_{33}, W_1, W_2, W_3 leads to a pruned trie with $2n + m$ nonleaf nodes, with $n = 3, m = 3$ Note that U_1, $\bar{U}_2$, $\bar{U}_3$ corresponds to $H = \{x, \bar{y}, \bar{z}\}$

LEMMA 3. *Let $F(I)$ have a pruned trie with access time no more than $h(2^h - 3n - m) + h(6n + 4m) + 3n(n + 1) + 2m(m + 1) + 4nm$. Let $n + m \geq 16$. Then the first h attributes tested in T are elements of N.*

PROOF. If the lemma is false, then an element, say A, of N is not one of the first h attributes tested. Consider a record β in L with a 1 entry for A. There must be another record β' such that β and β' agree in all bit positions except i, $1 \leq i \leq h$. If β' is also in L, then β and β' cannot be distinguished by the attributes in N that have been selected.

Since $3n + m$ h-bit integers are assigned to records in J ∪ K, at least $2^h - 6n - 2m$ records are distinguished no earlier than depth $h + 1$. The access time for T is therefore at least $A = (h + 1)(2^h - 6n - 2m)$. We show that $A > h(2^h - 3n - m) + h(6n + 4m) + 3n(n + 1) + 2m(m + 1) + 4mn$. Canceling terms and simplifying, we get

$$2^h > h(9n + 5m) + 3n(n + 1) + 2m(m + 1)4mn + 6n + 2m = \sigma. \qquad (1)$$

We prove (1) by determining an upper bound on σ and showing that 2^h is greater than the upper bound. For $n \geq 3$ and $m \geq 1$ we can derive $\sigma < (n + m)(9h + 6n + 6m)$. Since $h = 4 + \lceil 2 \log(n + m) \rceil$ for $n + m \geq 16$, $h < n + m$. Thus for $n + m \geq 16$, $9h + 6n + 6m < 15(n + m)$. We can therefore state that $\sigma < 15(n + m)^2$ since we are given $n + m \geq 16$.

In order to prove (1) we show that $2^h > 15\,(n + m)^2$. Since $h = 4 + \lceil 2 \log(n + m) \rceil$, $h \geq 4 + 2 \log(n + m)$ and

$$2^h \geq 2^4(n + m)^2 = 16(n + m)^2 > 15(n + m) > \sigma.$$

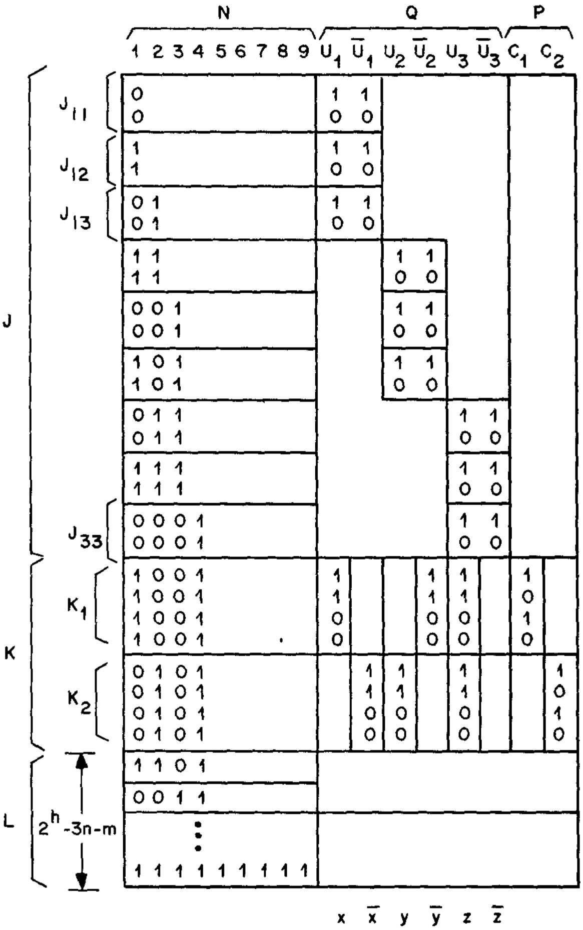
The lemma must therefore be true. □

THEOREM 4. *The least access time pruned trie problem for binary files is NP-complete*

PROOF. Given an instance I of SAT3, construct $F(I)$ as in Figure 8. We claim $F(I)$ has a pruned trie with access time no more than

$$h(2^h - 3n - m) + h(6n + 4m) + 3n(n + 1) + 2m(m + 1) + 4nm$$

if and only if I is satisfiable.



		N (1 2 3 4 5 6 7 8 9)	U_1	$\bar{U}_1$	U_2	$\bar{U}_2$	U_3	$\bar{U}_3$	C_1	C_2
J	J_{11}	0	1	1						
		0	0	0						
	J_{12}	1	1	1						
		1	0	0						
	J_{13}	0 1	1	1						
		0 1	0	0						
		1 1			1	1				
		1 1			0	0				
		0 0 1			1	1				
		0 0 1			0	0				
		1 0 1			1	1				
		1 0 1			0	0				
		0 1 1					1	1		
		0 1 1					0	0		
		1 1 1					1	1		
		1 1 1					0	0		
	J_{33}	0 0 0 1					1	1		
		0 0 0 1					0	0		
K	K_1	1 0 0 1	1			1	1		1	
		1 0 0 1	1			1	1		0	
		1 0 0 1	0			0	0		1	
		1 0 0 1	0			0	0		0	
	K_2	0 1 0 1		1	1		1			1
		0 1 0 1		1	1		1			0
		0 1 0 1		0	0		0			1
		0 1 0 1		0	0		0			0
L	2^h-3n-m	1 1 0 1								
		0 0 1 1								
		⋮								
		1 1 1 1 1 1 1 1 1								
			x	$\bar{x}$	y	$\bar{y}$	z	$\bar{z}$		

FIG 8 (Least access time pruned trie) Submatrix ⟨J ∪ K·Q ∪ P⟩ is very similar to Figure 4. The values assigned to set N are all possible h-bit binary numbers as shown The selection of all attributes from N produces a division of the records in ⟨J ∪ K Q ∪ P⟩ such that the proof of Theorem 1 applies

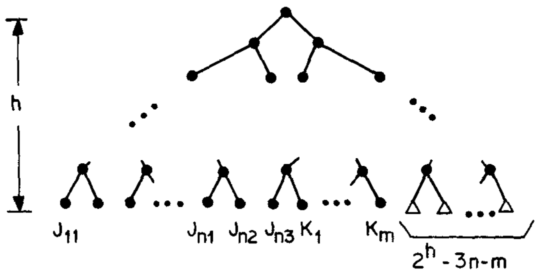


FIG 9 First h depths for a pruned trie for the file in Figure 8 Note the large number of leaves, $2^h - 3n - m$, at depth h

Suppose I is satisfiable Then construct T by first testing on all the attributes in group N. T will then have $2^h - 3n - m$ leaves at depth h accounting for the $h(2^h - 3n - m)$ term in the access time. The remaining selections proceed as in Figure 4 except that the $6n + 4m$ records in J ∪ K in Figure 8 will be distinguished h depths later than for the file in Figure 4 Thus the access time is $h(6n + 4m) + 3n(n + 1) + 2m(m + 1) + 4nm$.

Suppose for the second half that $F(I)$ does have a trie with access time no more than the bound Then, from Lemma 3, the first h attributes tested are elements of N. The trie must look like Figure 9 at depth h. The rest of the proof is similar to the proof of Theorem 1. □

REFERENCES

(Note Reference [11] is not cited in the text)

1 Aho, A V., Hopcroft, J E , and Ullman, J D *The Design and Analysis of Computer Algorithms* Addison-Wesley, Reading, Mass , 1974

2 Bayer, R , and McCreight, E Organization and maintenance of large ordered indices *Acta Informatica 1* (1972), 173-189

3 Comer, D E , and Sethi, R Complexity of trie index construction Extended abstract, Proc 17th Annual Symp on Foundations of Comptr Sci , Oct 1976, pp 197-207

4 Cook, S A The complexity of theorem-proving procedures Proc Third Annual ACM Symp on Theory of Computing, May 1971, pp 151-158

5 de la Briandais, R File searching using variable length keys Proc Western Joint Comptr Conf , IRE, New York, 1959, pp 295-298

6 de Maine, P A D , and Rotwitt, T Jr Storage optimization of tree structured files representing descriptor sets Proc ACM SIGFIDET Workshop on Data Description, Access and Control, Nov 1971, pp 207-217

7 Fredkin, E Trie memory *Comm ACM 3*, 9 (Sept 1960), 490-499

8 Hyafil, L., and Rivest, R Constructing optimal binary decision trees is NP-complete *Information Processing Letters 5*, 1 (May 1976), 15-17

9 Knuth, D E *The Art of Computer Programming, Vol 1 Fundamental Algorithms* Addison-Wesley, Reading, Mass , 1968

10 Knuth, D E *The Art of Computer Programming, Vol 3: Sorting and Searching* Addison-Wesley, Reading, Mass , 1973

11 Nievergelt, J Binary search trees and file organization *Computing Surveys 6,* 3 (Sept 1974), 195-207

12 Schkolnick, M Secondary index optimization ACM-SIGMOD Int Conf on Management of Data, San Jose, Calif , May 1975, pp. 186-192

13 Severance, D G Identifier search mechanisms: A survey and generalized model *Computing Surveys 6,* 3 (Sept 1974), 175-194

14 Sussenguth, E H Jr Use of tree structures for processing files *Comm ACM 6*, 5 (May 1963), 272-279

15 Yao, S B Tree structures construction using key densities ACM Annual Conf , Minneapolis, Minn , 1975, pp 337-340

RECEIVED MAY 1976, REVISED DECEMBER 1976