Tree Structures for Optimal Searching

L. E. STANFEL

Colorado State University, Fort Collins, Colorado*

ABSTRACT. It is shown that, owing to certain restrictions placed upon the set of admissible structures, some previous solutions have not characterized trees in which expected search time is minimized. The more general problem is shown to be a special case of a coding problem, which was previously formulated and solved as a linear integer programming problem, and in the special case of equally probable key requests is found to be solvable almost by inspection. Some remarks are given regarding the possibility of realizing a shorter computational procedure than would be expected from an integer programming algorithm, along with a comparison of results from the present method with those of the previous.

KEY WORDS AND PHRASES: file searching, information retrieval, trees, double chaining, minimum expected search time, variable length keys, integer programming

CR CATEGORIES: 3.70, 3.73, 3.74, 5.41

The purpose here is to discuss certain aspects of tree storage, including the material in E. H. Sussenguth's May 1963 paper [1]. To employ terminology consistent with that of Sussenguth, we call the set of all nodes lying on paths of unit length from a node i the *filial set* of node i. Nodes lying on paths of length $K - 1$ from a root of the tree are said to lie on the Kth level of the tree, roots lying on level 1. A sequence of edges leading from a root to a node on level $m + 1$ is called a path of length m. The structure of interest is illustrated by the typical example shown in Figure 1. Here, A, B, C, D are roots and lie on level 1; $\{E\}$ is the filial set of node F and lies on level 3; $\{H, I, J, K\}$ is the filial set of node G; $AMQP$ identifies a path of length 3; * denotes end of key.

Terminal nodes correspond to the pieces of information stored and intermediate nodes to components of the keys, a key being that which serves to label or identify the associated piece of information. For example, in an accounting system a record might consist of an account number and a name, the former serving as the key, the latter as the piece of information. In such a case the tree structure might appear as shown in Figure 2. Here, account 11456 has associated the name J. Farquard.

Referring to Figure 1, we describe the search procedure. Suppose it is desired to find that information associated with a given key. We compare the first key component to A, and if there is a match, compare the second key component to M. Otherwise, the first key component is compared to B. If the second key component matches M, the third component is compared to N; otherwise the second component is compared to F, etc. In short the search proceeds from left to right within a filial set, and when a match occurs begins with the first node in the filial set of the node which matched. To locate terminal node K in Figure 1, for example, nodes would

A portion of this work was completed at the University of Florida, Gainesville, Fla., under Project THEMIS, Contract ARO-D DAH-CO4-68C-0002.

* Department of Mechanical Engineering.

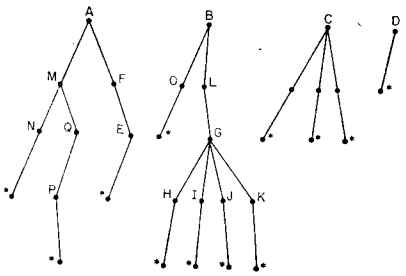
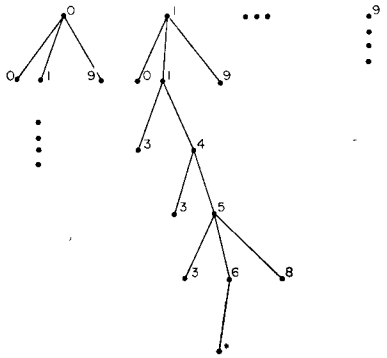


FIG. 1



J. Farquard

FIG. 2

be examined in the order $A, B, O, L, G, H, I, J, K$. We imagine that a two-dimensional chaining scheme exists which enables one to proceed both *within* a filial set and from a given node *into* its fiilial set.

Sussenguth's measure of search time is the number of chaining links traversed in the course of the search, where it is assumed that one link must be traversed to enter a filial set. Thus, in Figure 1, five links are required to locate P, four links for E, nine links for K, etc. Once in a filial set of S elements the expected number of links necessary to traverse is $\frac{1}{2}(S - 1)$, given that the node sought is equally likely to be any of those within the set.

Letting

$$s_i = \frac{\text{number of nodes on level } i}{\text{number of filial sets on level } i},$$

the expected search time (expected number of chaining links traversed) is then given [1] as

$$t = \tfrac{1}{2} \sum_{i=1}^{h} (s_i + 1). \tag{1}$$

We should observe, however, that eq. (1) holds only if all terminal nodes lie on level h of the tree.

At any rate, Sussenguth then proves that the minimization of t dictates that all s_i must be of identical size, 3.6 nodes, whereupon the optimal h may be calculated and is $\log_{3.6} N$, where N is the number of terminal nodes (number of items stored).

It was concluded, therefore [1], that "when it is possible freely to manipulate the keys, the key elements should be selected so that:

(i) all paths from a root to a leaf have the same length, h;

(ii) all filial sets have the same number of members, s; and

(iii) the common filial set size s is near 3.6 and h is $\log_s N$."

These results raise certain questions. In particular:

(1) In what sense is the solution obtained optimal?

(2) The problem of structuring the tree so as to minimize search time is integer in nature, so how are the noninteger results to be interpreted and implemented?

(3) How is the optimal tree structure to be deduced when the items of information have a nonuniform probability of being requested?

Regarding question (1), consider the case of 16 items. Sussenguth's results indicate that the optimal tree is that shown in Figure 3. For this arrangement, the expected search time, from eq. (1), is $\frac{1}{2} \cdot 2 \cdot 5 = 5$. The numbers near the terminal nodes in Figure 3 are the numbers of chaining links required to reach the corresponding nodes. The expected search time could therefore be calculated by summing those numbers and dividing by 16, i.e.

$$\bar{n} = \frac{14 + 18 + 22 + 26}{16} = \frac{80}{16} = 5.$$

Let us consider an alternative structure for a tree of 16 terminal nodes (see Figure 4). The numbers near the terminal nodes function as before. For this arrangement $\bar{n} = 77/16 = 4.81$.

Since we have discovered an expected search time less than 5 links, it is clear that in the usual sense Sussenguth's result definitely does not yield an optimal tree. It possesses, however, a certain "minimax" quality, for the subtrees of Figure 3 may be permuted in any fashion with $\bar{n}$ remaining invariant. The structure in Figure 4, on the contrary, does not possess this attribute. For, consider Figure 5. Here, $\bar{n} = 5.5$ chaining links.

If one is interested in storing a file in tree format, however, with minimization of expected search time the objective, the effect upon search time of perturbations of this kind is not the deciding factor. It is clear that the assumption of a common level for all terminal nodes in an optimal tree is invalid and has led to a nonoptimal result.

In [1] it is apparently assumed that all possible input keys have identical length. Free manipulation of the keys is taken to mean that there exists a 1:1 transformation $T:K \rightarrow K'(h)$, where K is the set of all possible keys with which the system will have to deal and $K'(h)$ is a set of strings of symbols of length h, where h is determined as a value in a certain minimization problem. The system must be capable of determining $k'(h)$ from a given k (including those k it has not previously seen), or else a translation must take place before input.

In this paper, free manipulation of keys is taken to mean that there exists a 1:1 transformation $C:K \rightarrow L$, where L is a set of symbol strings of variable length, the particular lengths being integers obtained from a minimization problem (to be formulated below).

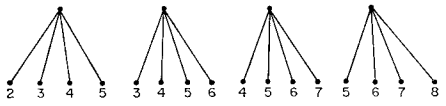


FIG. 3

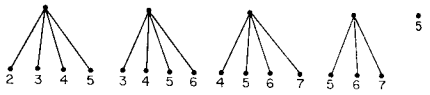


FIG. 4

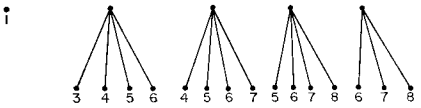


FIG. 5

This relaxation of restrictions on the set of admissible solutions enables one to improve upon the result obtained by employing (i)–(iii) in every instance, although key translation may become more complicated in cases. (Key translation time in any situation should rightfully become a part of the expected search time. If this were a sufficiently costly process, one might prefer to allow *no* manipulation of keys, although this modification of objective function will not be carried out here.)

Let us now consider question (2). Since we are interested in finding an optimal tree, we must be concerned with quantities such as the number of nodes in a filial set, the number of chaining links required to reach a given terminal node, etc. These quantities are all integers, of course, and any description of an optimal tree must also be in integer terms.

If this optimization problem is mathematically formulated in the most straightforward fashion, one obtains an integer programming problem, which is quite large and unwieldy, and further complicated by certain nonlinearities within the constraints. Without giving this formulation, let us pursue a less straightforward approach which results in a formulation much more amenable to solution. Motivated by considerations of practicality, we do choose to add one further constraint to the formulation. It seems reasonable to expect that the file designer has a priori knowledge of the number of symbols available for use as key elements—the keys may have to be binary; it may be that the keys are to be strings composed of the elements $\{0, 1, 2, \cdots, 9\}$; or, the key-coding alphabet may be the set of Latin letters; etc. Therefore, we assume we are given a maximum number of possible key elements, allowing that if it is possible to do better with less than all of them, then we expect our solution procedure to so advise.

In a 1960 paper [2], Richard Karp treated the problem of constructing minimum redundancy prefix encodings in the event the coding symbols have rational, but perhaps unequal, costs—a set of circumstances which preclude the use of Huffman's well-known procedure. Karp was able to cast this problem into the following form.

Minimize

$$\sum_{i=1}^{n} \sum_{j=1}^{m} jP_iY_{ij}$$

subject to

$$\sum_{i=1}^{n} Y_{ij} + b_j \leq \sum_{K=1}^{r} b_{j-c_K}, \quad j = 1, \cdots, m, \tag{P1}$$

$$\sum_{j=1}^{m} Y_{ij} = 1, \qquad i = 1, \cdots, n,$$

with $b_0 = 1$, $b_q = 0$ for $q < 0$ and Y_{ij}, b_j nonnegative integers.

To explain the remainder of the notation in Karp's problem, n was the number of code words; $\{P_i\}$ was a set of stationary probabilities where Pr {code word i is sent} $= P_i$; the costs of the r available code symbols were $C_1, C_2, \cdots, C_r$ (integers); m was an upper bound on the cost of code words (m an integer), and b_{j-c_K} was the number of prefixes of cost j which terminate with the Kth symbol. Karp was able to reduce the number of variables in the problem by introducing certain bounds and solving a sequence (perhaps only one) of problems of the form (P1), each with some of its variables fixed. The result was that problems with a larger number of variables

could still be effectively treated. Some computational results are reported in his paper. The solution procedure was Gomory's all-integer cutting plane method [3] implemented on the IBM 704 computer.

We now show that the problem of constructing an optimal doubly chained tree is but a special case of Karp's problem and one, therefore, for which a very well-defined solution procedure already exists. It is assumed that there are r symbols available as key components and that the number of chaining links necessary to reach a terminal node must be no greater than some given positive integer m. Now consider a prefix encoding of n words over an alphabet of r characters, the symbols 1, 2, $\cdots$, r for example. The graph of such an encoding would appear as in Figure 6, where symbols are assigned in ascending order within a filial set. Here the code words are {11, 121, 122, $\cdots$, $r1$, $r2$, $r31$}.

Suppose in such a coding tree we let the cost of the symbol $i = i$, $i = 1, \cdots, r$. Then *the cost of a code word is exactly the number of chaining links required to reach the terminal node associated with that code word*, if one searches through the structure as if it were a doubly chained tree. This is clear from Figure 6, as well as from the construction of the tree.

Now if an encoding had, in any of the filial sets in its tree, the symbols in an order other than 1, 2, $\cdots$, r, then it would not always be the case that the cost of a code word equals the number of chaining links required to reach the associated terminal node. But, clearly, every filial set in the tree of an *optimal* encoding will have the symbols ordered in the desired fashion—otherwise, the encoding would not be optimal. Thus, if we can construct a minimum redundancy prefix encoding for n words over an r-symbol alphabet, where the cost of symbol i is i, $i = 1, \cdots, r$, then we have also found that tree structure whose expected search time is minimized.

We emphasize that the tree structures obtained will be optimal over all those trees in which no key prefixes another, since Karp's approach locates optimal prefix codes.

Now in the event the items of information have equal probabilities of being sought, we may simplify problem (P1) and obtain (P2):

Minimize

$$\sum_{j=1}^{m} Z_j j$$

subject to

$$\begin{aligned}
&Z_1 + b_1 \leq b_0 = 1,\\
&Z_2 + b_2 \leq b_1 + b_0,\\
&Z_3 + b_3 \leq b_2 + b_1 + b_0,\\
&Z_j + b_j \leq \sum_{i=1}^{r} b_{j-i},\\
&Z_m + b_m \leq \sum_{i=1}^{r} b_{m-i},\\
&\sum_{j=1}^{m} Z_j = n,
\end{aligned} \tag{P2}$$

and b_j, Z_j are nonnegative integers, $j = 1, \cdots, m$.

(P2) remains a linear, integer programming problem, but now we have a problem that can be solved virtually by inspection.

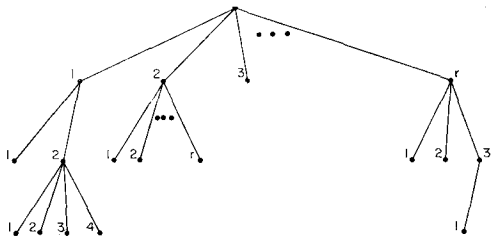


FIG. 6

Let us proceed to give an algorithm for solving (P2), illustrate its use by example, and then prove that it is effective. In what follows we speak in terms of code words and codes rather than keys, trees, etc.

Denote by b_K^* the maximum value that the variable b_K may assume. For example, $b_1^* = 1$, $b_2^* = 2$, $\cdots$, $b_j^* = \sum_{i=1}^{r} b_{j-i}^*$, $\cdots$. Let us determine two costs, $K + 1$ and $K + 2$, as follows. We want $n > b_{K+1}^*$ and $(n - b_{K+1}^*) \leq b_{K+2}^* - b_{K+1}^*$. Clearly, from the nature of the b_j^*, $(K + 1)$ and $(K + 2)$ are uniquely determined. At this point we let b_{K+1}^* words assume cost $(K + 1)$ and $(n - b_{K+1}^*)$ words assume cost $(K + 2)$. The constraints of (P2) are satisfied with $Z_1 = Z_2 = \cdots = Z_K = Z_{K+3} = \cdots = Z_m = 0$; $b_j = b_j^*$, $\cdots$, $j = 1, \cdots, K$, $b_{K+1} = 0$, $b_{K+2} = 0$. Next we take b_K^* of the words of cost $(K + 1)$ and let them each assume cost K. Now, $b_K = 0$, $Z_K = b_K^*$, $Z_{K+1} = b_{K+1}^* - b_K^*$, $b_{K+1} = 0$, and if the $(K + 2)$-nd constraint is still satisfied, we stop. Otherwise, we increase Z_{K+1} by reducing Z_K and increasing b_K simultaneously until the $(K + 2)$-nd constraint is just satisfied ($b_{K+2} = 0$).

The claim is that the resulting code is optimal. First, we give two examples.

Let $r = 5$, $n = 11$, $m = 7$. (If m is not formally specified in the tree problem the procedure does not suffer.) We first generate the b_j^*; $b_1^* = 1$, $b_2^* = 2$, $b_3^* = 4$, $b_4^* = 8$, $b_5^* = 16$, and this will suffice. According to the algorithm we allocate 8 words to cost 4, 3 words to cost 5. The pertinent constraints of (P2) then appear:

$$Z_4 + b_4 \leq b_3 + b_2 + b_1 + b_0,$$

$$Z_5 + b_5 \leq b_4 + b_3 + b_2 + b_1 + b_0,$$

or

$$8 + 0 = 4 + 2 + 1 + 1 = 8,$$

$$3 + b_5 \leq 4 + 2 + 1 + 1 = 8,$$

and

$$0 \leq b_5 \leq 5.$$

Next, we allocate 4 words to cost 3, keeping 4 of cost 4, 3 of cost 5.

$$Z_3 + b_3 \leq b_2 + b_1 + b_0.$$

But if $Z_3 = 4$, $b_3 = 0$, since $b_3^* = 4$. Next we have

$$Z_4 + b_4 \leq b_3 + b_2 + b_1 + b_0,$$

$$4 + 0 = 0 + 2 + 1 + 1 = 4.$$

Finally,

$$Z_5 + b_5 \leq b_4 + b_3 + b_2 + b_1 + b_0,$$

$$3 + b_5 \leq 0 + 0 + 2 + 1 + 1.$$

All constraints are satisfied, and we stop.

Next, consider the case $r = 5$, $n = 14$, $m = 7$. A few of the b_j^* are, again, $b_1^* = 1$, $b_2^* = 2$, $b_3^* = 4$, $b_4^* = 8$, $b_5^* = 16$.

The algorithm states that we first allocate 8 words to cost 4, 6 words to cost 5. Next we allocate 4 words to cost 3, keeping 4 of cost 4, 6 of cost 5. Examining the constraints of (P2) then, we find:

$$Z_3 + b_3 \leq b_2 + b_1 + b_0$$

or

$$4 + b_3 \leq 4,$$

so

$$b_3 = 0.$$

$$Z_4 + b_4 \leq b_3 + b_2 + b_1 + b_0 = b_2 + b_1 + b_0$$

or

$$4 + b_4 \leq 4,$$

so

$$b_4 = 0.$$

$$Z_5 + b_5 \leq b_4 + b_3 + b_2 + b_1 + b_0 = b_2 + b_1 + b_0,$$

$$6 + b_5 \leq 4,$$

and the fifth constraint cannot be satisfied.

If, however, we decrease Z_3 by 2, b_3 may increase by 2. Then Z_4 may be increased by 2, the fourth constraint remains satisfied, and the fifth constraint is then just satisfied with $b_5 = 0$, and the resulting code has 2 words of cost 3, 6 of cost 4, 6 of cost 5.

The process of determining the structure of the code (or tree) is entirely simple, regardless of the magnitudes of n, r, m. Once the b_j^* are generated, the problem is essentially solved by inspection. For large values of the parameters n, r, m, a computer could easily generate the required numbers—certainly both programming time and execution time would be negligible for this procedure.

The final task here is to demonstrate that the result obtained does constitute an optimal solution to the problem.

We may observe that the result is optimal among all those codes with minimum-cost words having cost K or greater, since it has that number of words of cost K which allows the number of words of cost $K + 2$ to be minimized and has no words of cost greater than $K + 2$.

The next step is to establish that introducing words of cost less than K cannot improve upon the result already obtained.

It is instructive to first observe what occurs when a word of cost $(K - 1)$ is intro-

duced into our result. Suppose we select a word of cost K and let it assume cost $K - 1$. Immediately, b_{K-1} must decrease by 1, b_K may remain unchanged, but b_{K+1} must decrease by 1. But b_{K+1} was zero, so a word of cost $(K + 1)$ must assume cost $(K + 2)$ at least, and, at best, we realize no improvement.

Suppose we select a word of cost $(K + 1)$ to assume cost $(K - 1)$. Again, b_{K-1} is reduced by 1 and, as a result, one of Z_K, b_K must be reduced by 1 also. (If, in fact, $b_K \neq 0$. It may be that there is no choice.) Suppose b_K is reduced by 1. Then, since $b_{K+1} = 0$ and Z_{K+1} was already reduced by 1, Z_{K+1} must be reduced one more, and the corresponding word cannot assume cost $(K + 2)$, because the $(K + 2)$-nd constraint is just satisfied with $b_{K+2} = 0$, which in turn results from the fact that b_K was nonzero in the first place. So the cost of the displaced word increases by 2, and since the right side of the $(K + 2)$-nd constraint involves b_K, which was reduced by 1, at least one other word of cost $(K + 2)$ must assume cost $(K + 3)$, and consequently the average word cost has been increased. In similar fashion we would find that decreasing Z_K, rather than b_K, yields the same conclusion, and the same sort of arguments will show that introducing a word of cost $(K - 1)$ at the expense of Z_{K+2} also worsens the average word cost.

So no improvement is possible upon introducing one new word of cost $K - 1$, and clearly the same conclusion is unavoidable if more words of cost $K - 1$ are introduced.

More generally, suppose there exist codes with minimum cost $K - p$, $p \geq 1$, which give lower average cost than our originally obtained code, W. Denote the best code of this set W_1, and let us proceed to operate upon W_1, which has minimum-cost word with cost $K - p$, $p \geq 1$. Suppose we decrease Z_{K-p} by 1. Then b_{K-p} may be increased by 1; b_{K-p+1}, by 1; b_{K-p+2}, by 2; b_{K-p+3}, by at least 3 (when $r = 2$, the possible increase would be 3); $\cdots$; b_{K-p+j}, by at least F_j, where $F_j = j$th Fibonacci number. ($F_0 = 1$, $F_1 = 1$, $F_i = F_{i-1} + F_{i-2}$, $i \geq 2$.)

We find, in particular, that b_K may be increased by at least F_p. We propose to allow our deleted word of cost $K - p$ to assume cost K, which allows b_K or consequently Z_K to be increased at least $F_p - 1$, at the expense of costlier words.

With each of these $F_p - 1$ remaining words we can save at least one unit of cost. Now Z_{K+1} may also be increased at least 1 at the expense of costlier words, since b_{K-1} has increased. Suppose we save just one more unit of cost.

We have, then, reduced total cost $F_p - 1 + 1 = F_p$ and increased it p. But $F_p \geq p$, all p. Thus we have not increased the total cost. Continuing in this fashion, increasing the cost of words having cost less than K by allowing them to assume cost K, we either reconstruct our original code W and find that it gives as good a result as the hypothesized one, W_1, or we discover strict improvement at some point along the way. In either case, we contradict the assumption that W_1 gives an average word length smaller than any other. Consequently, the code W must be optimal.

Conclusions

We were able to formulate the problem as a linear integer programming problem, solvable by a variety of existing techniques, but nevertheless, for a large number of variables and constraints, not a problem for which extremely efficient algorithms are available. However, in the event of a uniform probability distribution, probably the most important case, the problem has been found to be quite easily solved—almost by inspection, in fact.

For the general probability distribution, no such attractive result can be expected, but there is a possibility that computational savings might be realized even in this more general case. If we write out the constraints of (P1) we notice that the first $m \cdot n$ columns are precisely those of an ordinary transportation problem. Now this array itself is unimodular; that is, every minor has value 0 or ± 1, and this property *guarantees* an optimal solution *in integers*, given that the constants in the constraints are integers. If our entire matrix of coefficients in (P1) had this property, then we could remove the integer requirements, solve (P1) as a *linear programming problem*, and obtain an optimal solution in integers.

Unfortunately the array one finds in (P1)—after converting to a system of equalities—is not unimodular, but for small m, n it seems to be the case that many nonsingular $(m + n) \times (m + n)$ arrays, which qualify as bases for the linear programming solution procedure, do have determinant ± 1. Therefore, we may expect optimal solutions obtained for the linear programming problem to be integer more frequently than in general. Further, it would be more likely to find an alternative optimal solution in integers than in general. Alternative optima are quite easily generated once the linear programming problem has been solved, and exploring these possibilities before entering the actual integer programming procedure (it is typical of integer programming algorithms to first solve the problem sans integer requirements) might prove profitable. These latter remarks are, of course, conjecture, their worth being dependent upon the frequency with which sets of $m + n$ linearly independent columns from the matrix of constraint coefficients in (P1) have determinant ± 1.

It may be of interest to compare a few sample results with variable length keys with those of the fixed length keys. In Table I, for the fixed length keys, we let the average filial set size be 3.6, as required by (iii), and computed h to be the smallest integer satisfying $h \geq \log_{3.6} N$, from (iii), since h must be an integer. The numbers in columns 2–5 are the average search times, measured in numbers of chaining links.

Regarding Table I, several remarks are appropriate. First, the values of N chosen are of no particular significance—it was desirable to represent a range of magnitudes, and the values chosen accomplish that.

The numbers of symbols for which we computed expected search time in the variable length case were chosen only to give a small range of values which included the average filial set size of 3.6 for the fixed length keys.

It should be realized that it is actually inequitable to compare the fixed length results with the variable length for two and three symbols, since if the common filial set size is to be 3.6, there must be more than three symbols available. It is the case, however, that even the three-symbol result is superior to the fixed length, giving better results for all N. The further improvement for four or five symbols is apparent.

TABLE I

	$N = 50$	$N = 100$	$N = 1000$	$N = 10{,}000$
Fixed length	9.2	9.2	13.8	18.4
Variable—2 symbols	8.2	9.7	14.4	19.2
Variable—3 symbols	6.6	7.8	11.8	15.6
Variable—4 symbols	6.3	7.4	10.8	14.4
Variable—5 symbols	6.0	7.2	10.6	13.9

It should also be observed that the differences obtained are not trifling—a significant improvement is achieved when the keys may have variable length, and if there were more symbols than five available, the results would appear even more favorable.

It must be pointed out that the simplest problem of the class described here, namely, that in which the probability distribution of inquiries is uniform, with the further assumption *that the number of available key symbols is unlimited*, has been solved in quite different fashion [4]. As mentioned previously, an upper bound on the number of available symbols is a realistic constraint.

It is felt that the present approach is preferable to that in [4], since the strong relationship between the search problem and the coding problem is here evident—it is, in fact, exploited. Further, the fact that such trees as are obtained are optimal in a *prefix* sense is clear from the coding context; otherwise, the distinction is not so apparent.

A reviewer has noted the paper by Arora and Dent [5], and has suggested that reference to it be made here. The two problems and the assumptions made in the two differ in several significant respects.

In the context of [5], incoming items are already labeled in such a way as to constrain the tree eventually created; there are essentially but two key symbols available for use; there is no prefix requirement on keys; each vertex requires the storage of three addresses (owing to the prefixing of keys by others).

In the present context, however, incoming items are essentially unlabeled and the order of arrival has no effect upon the structure of the tree eventually obtained; the number of available symbols is the parameter which affects the structure of the tree and this number is utilized to greatest advantage; we have imposed a prefix requirement, which in fact increases the expected search time [6], and we have a two-address system.

Thus, although both methods involve constructing and searching a file which possesses a tree form, the circumstances within which they are appropriate are quite different.

REFERENCES

1. SUSSENGUTH, E. H. Use of tree structures for processing files. *Comm. ACM 6*, 5 (May 1963), 272–279.
2. KARP, R. M. Minimum redundancy coding for the discrete, noiseless channel. *IRE Trans. IT-7* (1961), 27–35.
3. GOMORY, R. E. An algorithm for integer solutions to linear programs. In *Recent Advances in Mathematical Programming*, Graves, R. L., and Wolfe, P. (Eds.), McGraw-Hill, New York, 1963, pp. 269–302. (First made known in 1958.)
4. PATT, Y. Variable length tree structures having minimum average search time. *Comm. ACM 12*, 2 (1969), 72–76.
5. ARORA, S. R., AND DENT, W. T. Randomized binary search technique. *Comm. ACM 12*, 2 (1969), 77–80.
6. STANFEL, L. E. A comment on optimal tree structures. *Comm. ACM 12*, 10 (Oct. 1969), 582.

RECEIVED MARCH, 1969; REVISED DECEMBER, 1969