SIAM J. COMPUT.
Vol. 5, No. 1, March 1976

OPTIMAL ALPHABETIC TREES*

ALON ITAI†

Abstract. An algorithm of Knuth for finding an optimal binary tree is extended in several directions to solve related problems. The first case considered is restricting the depth of the tree by some predetermined integer K, and a Kn^2 algorithm is given. Next, for trees of degree σ, rather than binary trees, $Kn^2 \log \sigma$ and $n^2 \log \sigma$ algorithms are found for the restricted and nonrestricted cases, respectively. For alphabetic trees with letters of unequal cost, a $\sigma^2 n^2$ algorithm is proposed. We conclude with a comparison of alphabetic and nonalphabetic trees and their respective complexities.

Key words. algorithms, alphabetic trees, binary trees, optimal trees, probabilistic search, variable-length codes

1. Introduction. When constructing a code, it is often necessary to minimize the average message length. There is a natural correspondence between a binary prefix code and a binary tree, associating with every leaf a codeword. Assuming that every source word has a fixed probability of occurring, the length of an average message corresponds to the weighted path length of the tree. Let $(w_1, \cdots, w_n)$ denote the weights of the codewords and $l(w_i)$ the length of the path from the root to the weight w_i. The weighted path length (cost) is defined as $WT = \sum_{i=1}^{n} w_i l(w_i)$.

In real time applications, it may be desirable to restrict the length of every codeword. In the tree representation, this restriction is expressed by limiting the length of any path from the root of the tree to a leaf by some integer K.

Similar problems arise when organizing files. Each entry of the file has a certain probability of being requested. There is an order between the entries of the file which must be conserved. No search can take longer than a prespecified maximum. Our aim is to construct a file which fulfills all the requirements and minimizes average search time.

The file is represented as a binary tree with the entries at the nodes. Our aim is to find, among all trees which satisfy the restriction that no path is longer than a given integer K and which preserve the order of the file entries, one with minimum cost.

In the paper, we show that an algorithm of Garey [1], which is an extension of an algorithm of Knuth [4], can be used to solve several problems of this nature.

The running time is of order Kn^2, where n is the number of nodes, K the maximal depth.

In § 7 we show how to extend the results from binary trees to σ-ary trees (i.e., trees in which each node may have up to σ sons). For this problem, we give $n^2K \log \sigma$ and $n^2 \log \sigma$ algorithms for the restricted and nonrestricted cases, respectively.

* Received by the editors July 30, 1974, and in revised form December 30, 1974.

† Department of Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel.

Section 8 deals with alphabetic trees with letter of unequal cost. For this problem a $\sigma^2 n^2$ algorithm is proposed. Finally, in the last section, we compare the problem of alphabetic and nonalphabetic trees and their respective complexities.

2. Definitions. A *binary tree* consists of a distinguished node, called *root*, and right and left subtrees of the root. A *binary full tree* consists of a root and left and right subtrees, either both empty or both binary full trees. Nodes occurring in the two subtrees are called *descendants* of the root. All nodes having a given node as a descendant are *ancestors* of that node. If j is the root of a subtree of i, then i is the *father* of j and j is i's *son*. Nodes which have no descendants are called *leaves*. Nodes which are not leaves are called *internal nodes*. From the root there is a unique path to every node i. The length of that path is the number of nodes in the path minus 1 and is called the *depth of node i*, denoted by l_i.

Traversing a tree in postorder (symmetric order) [3] (visit the left subtree, visit the root, visit the right subtree), imposes a full order relation on the nodes. We say that node i is left of node j $(i<j)$ if i precedes j in that order.

An *alphabetic tree* for the sequence of nonnegative weights $(w_1, \cdots, w_n)$ is a binary tree with n leaves, each of which is associated with a weight, such that the leaf of w_i is left of the leaf of w_{i+1} for $i=1, \cdots, n-1$.

The *weighted path length* or *cost* of an alphabetic tree T is given by

$$WT=\sum_{i=1}^{n} w_i l_i,$$

where l_i is the depth of the leaf associated with the weight w_i.

Given a positive integer K a *K restricted alphabetic tree* is an alphabetic tree in which no node has depth greater than K. A K restricted alphabetic tree is optimal if it has minimum cost. In the sequel, we shall use the phrase *optimal tree* in lieu of the cumbersome optimal K restricted alphabetic tree.

3. A dynamic programming solution. We first notice that an optimal tree is a binary full tree (each internal node has exactly two sons), except perhaps when zero weights are involved. However, in this case, there exists an optimal tree which is a full binary tree. Henceforth, we shall assume that all optimal trees are full binary trees.

Let T be an optimal tree, v an internal node of T of depth l_v; then the subtree of T with root v is an optimal $(K-l_v)$ restricted alphabetic tree for its own sequence of weights.

Let $[i, j, k]$, $1 \leqq i \leqq j \leqq n$ and $0 \leqq k \leqq K$, denote the subproblem of finding an optimal tree of depth at most k for $(w_i, w_{i-1}, \cdots, w_j)$. If $k < \lfloor \log_2 (j-i+1) \rfloor$, then no solution exists, and we may set $WT[i, j, k]=\infty$. Otherwise set

$$WT[i, i, k]=0, \qquad i=1, \cdots, n,$$

$$(1) \qquad WT[i, i, k]=\sum_{r=i}^{j} w_r+\min_{i\leqq b<j}\{WT[i, b, k-1]+WT[b+1, j, k-1]\},$$

$$i=1, \cdots, n-1, j=i+1, \cdots, n.$$

The value b_0, for which (1) produces the minimum, is the last node (in postorder) of the left subtree and is called the *breakpoint*. If b_0 is a breakpoint for $[i, j, k]$, we can construct an optimal tree for $[i, j, k]$ by constructing an optimal tree for $[i, b_0, k-1]$ and $[b_0+1, j, k-1]$ and combining these two trees. Solving $[i, j, k]$ for all k, i, j (looping first on $k = 1, \cdots, K$, then on $p = j-1 = 1, \cdots, n-1$, then on $i = 1, \cdots, n-p$) results in an optimal tree for $[1, n, K]$ the original problem.

After finding the minimum cost, we may reconstruct the tree had we saved the breakpoints b_0 for every $[i, j, k]$.

The resulting algorithm requires execution time proportional to Kn^3. Gilbert and Moore [2] proposed a similar algorithm for trees with no restriction on maximal depth. See that paper for further description and analysis of the algorithm.

4. An improved algorithm. We may improve the previous algorithm by observing the following phenomena.

THEOREM 1. *Let b_0 be the breakpoint of an optimal tree for* $[1, n, k]$. *Then there exists an optimal tree for* $[1, n+1, k]$ *whose breakpoint b'_0 satisfies $b'_0 \geqq b_0$.*

Let $b[i, j, k]$ denote a breakpoint of the subproblem $[i, j, k]$ and $b^l[i, j, k]$ the breakpoint of the optimal tree with the smallest breakpoint *leftmost tree.*

THEOREM 2. *There exist optimal trees such that*:

(a) $b[i, j-1, k] \leqq b[i, j, k] \leqq b[i+1, j, k]$. *Moreover, the leftmost trees satisfy* (a); *in other words,*

(b) $b^l[i, j-1, k] \leqq b^l[i, j, k] \leqq b^l[i+1, j, k]$.

We shall prove the theorems in the next section. Let us first see how we can use these results to obtain an improved algorithm. For fixed k and fixed $p = j-1$, the smallest breakpoint of $[i, j, k]$-$b^l[i, j, k]$ lies between $b^l[i, j-1, k]$ and $b^l[i+1, j, k]$ (Theorem 2(b)). Therefore, we need only consider $b^l[i+1, j, k] - b^l[i, j, k]$ possible breakpoints. Summing over i and substituting $j = i+p$, we find that

$$\sum_{i=1}^{n-p} (1 + b^l[i+1, i+p, k] - b^l[i, i-1+p, k])$$
$$= (n-p) + b^l[n-p+1, n, k] - b^l[1, p, k] \leqq 2n.$$

The improved algorithm determines $b^l[i, j, k]$ and $WT[i, j, k]$ using previously obtained values. The outer loops on k and p introduce a factor of nK. The whole algorithm takes time proportional to n^2K.

Note that the results of Theorem 2(a) are not sufficient since there may be optimal trees which satisfy $b[i, j-1, k] > b[i+1, j, k]$. We overcome this difficulty by insisting on the smallest breakpoint. This refinement does not increase the difficulty of implementing the algorithm and is usually incorporated for the sake of programming convenience [4, p. 22].

Knuth [4] first proposed this algorithm for the unrestricted case. Garey [1] extended it to the restricted case when $w_1 \leqq w_2 \leqq \cdots \leqq w_n$. Here we show that this additional requirement is superfluous.

5. Proof of Theorems 1 and 2. Let T, T' be trees of depth not exceeding K. Let a be a node of T, a' a node of T'. Both T and T' are subgraphs of a binary full tree and can be compared by the "left of" relation; i.e., a left of a' ($a < a'$), $a = a'$ or a right of a' ($a' < a$). Let x_i denote the position of weight w_i in the tree T, x'_i the position of w_i in T'. In what follows, we shall assume $k \geqq \log_2 (n+1)$. First we notice that by adding to the right a new node of weight $w_{n+1} = 0$ w_n moves to the left.

LEMMA 3. *Let $T = T[1, n, k]$ be an optimal tree for weights $(w_1, \cdots, w_n)$. There exists an optimal tree $T' = T'[1, n+1, k]$ for weights $(w_1, \cdots, w_n, 0)$ such that $x'_n < x_n$.*

Proof. Case 1. x_n is of maximal depth. (i.e., $l_n = k$). In this case, $x'_{n+1} \leqq x_n$ since $x'_n < x'_{n+i}$ by transitivity $x'_n < x_n$.

Case 2. x_n is not of maximal depth (i.e., $l_n < k$). In this case, we may obtain an optimal tree by replacing w_n by a node whose two sons are w_n, w_{n+1}. The resulting tree T' satisfies $WT' = WT + w_n$. To show that T' is optimal, assume the existence of a tree $T^2 = T^2[1, n+1, k]$ which costs less than T', $WT^2 < WT'$. We may construct a tree $T^3 = T^3[1, n, k]$ by deleting w_{n+1} from T^2. This would enable us to decrease the depth of w_n. We then have

$$WT^3 \leqq WT^2 - w_n < (WT + w_n) - w_n = WT,$$

which contradicts the optimality of T. Q.E.D.

Next we note that w_n does not have to move right when w_{n+1} increases.

LEMMA 4. *Let $T = T[1, n+1, k]$ be an optimal tree for weights $(w_1, \cdots, w_n, w_{n+1})$. There exists an optimal tree $T' = T'[1, n+1, k]$ for weights $(w_1, \cdots, w_n, w'_{n+1})$, $w'_{n+1} > w_{n+1}$, such that $x'_n \leqq x_n$.*

Proof. If the tree T remains optimal for weights $(w_1, \cdots, w_n, w_{n+1})$, we are done since we can choose $T' = T$. Otherwise, let T' be any optimal tree for $(w_1, \cdots, w'_{n+1})$. Consider the cost of T and T' as functions of the $(n+1)$st weight, w: $WT(w) = w_1 l_1 + \cdots + w_n l_n + w l_{n+1} = C + l_{n+1} w$, $WT'(w) = w_1 l'_1 + \cdots + w_n l'_n + w l_{n+1} = C' + l'_{n+1} w$. The difference in cost $D(w) = WT'(w) - WT(w) = C' - C + (l'_{n+1} - l_{n+1}) w$ is a linear function of w, and we know that $D(w'_{n+1}) < 0 \leqq D(w_{n+1})$.

It follows that $l'_{n+1} < l_{n+1}$. In particular, the father z of x_{n+1} is a node of the right subtree of the father z' of x'_{n+1}. Since x'_n is in the left subtree of z' and x_n is a descendant of z, we have $x'_n < x_n$. In both cases we find that $x'_n \leqq x_n$. Q.E.D.

If we add a new weight w_{n+1} to the right, the weight w_n moves left, because we can do this in two stages: First add a node of weight $w_{n+1} = 0$. This enables us to move w_n to the left. Then increase the weight of the node. This may be done without moving w_n to the right. Thus we have the following corollary.

COROLLARY 5. *If $T = T[1, n, k]$ is an optimal tree for $(w_1, \cdots, w_n)$, there exists an optimal tree $T' = T'[1, n+1, k]$ for $(w_1, \cdots, w_{n+1})$ such that $x'_n < x_n$.*

We wish to show that by adding a new node, we form a tree in which none of the previous weights moves right. It has already been shown that we can construct an optimal tree in which w_n moves left. We shall now show that this requirement may be extended, and we can find an optimal tree in which no weight moves to the right.

Some notation follows: For Q a binary tree, let SQ denote the set consisting of all indices of the weights of Q. WQ as before denotes the cost of Q.

LEMMA 6. *Let $T = T[1, n, k]$ be an optimal tree for $(w_1, \cdots, w_n)$. There exists an optimal tree $T' = T'[1, n+1, k]$ for $(w_1, \cdots, w_{n+1})$ such that $x'_i \leqq x_i$, $i = 1, \cdots, n$.*

Proof. Assuming to the contrary that no such optimal tree exists, let T' be an optimal tree for weights $(w_1, \cdots, w_{n+1})$ in which some weights move right. Let w_{j-1} be the last weight which moves right. ($x_{j-1} < x'_{j-1}$ but $x'_i \leqq x_i$ for $i = j, \cdots, n$.) By Corollary 5, we may assume $j \leqq n$. Let z be the deepest common ancestor of x_{j-1} and x_j (i.e., a node z which is an ancestor of both x'_{j-1} and x'_j, and no descendant of z has this property).

Comparing x'_{j-1} and x'_j to z, there are three possibilities:

(a) w_{j-1} and w_j do not pass z ($x_{j-1} < x'_{j-1} < z < x'_j \leqq x_j$);

(b) w_{j-1} passes z to the right ($x_{j-1} < z < x'_{j-1} < x'_j \leqq x_j$);

(c) w_j passes z to the left ($x_{j-1} < x'_{j-1} < x'_j < z < x_j$).

(We cannot have x'_{j-1} (or x'_j) equal to z since this would force x'_j to move right of x_j (or x'_{j-1} to move left of x_{j-1}).) z must be a node of T', since otherwise x'_{j-1} would be forced right of z, and so would all its descendants, in particular $x_j < x'_{j-1}$. This contradicts the hypothesis that $x'_{j-1} < x'_j \leqq x_j$.

Let $L(z)$ be the partial tree containing all the nodes left of z in T and the paths connecting them to the root of T. Define $R(z)$ in the same fashion on the nodes right of z in T. $L'(z)$ and $R'(z)$ are the analogous partial trees of T'.

Case (a). w_{j-1} and w_j do not pass z. $SL(z) = \{1, \cdots, j-1\} = SL'(z)$ (the leaves of $L(z)$). Construct T^2 from $L(z)$ and $R'(z)$. The leaves of T^2 are $\{1, \cdots, n+1\}$ the same as of T'.

$$WT^2 = WL(z) + WR'(z) \geqq WT' = WL'(z) + WR'(z).$$

If $WT^2 = WT'$ we have constructed, contrary to the hypothesis, an optimal tree for $(w_1, \cdots, w_n, w_{n+1})$ in which no weights move right.

Therefore $WL(z) > WL'(z)$. Construct T^3 from $L'(z)$ and $R(z)$. $WT^3 = WL'(z) + WR(z) < WL(z) + WR(z) = WT$. T^3 consists of the same weights as T, so the last inequality contradicts the optimality of T.

Case (b). w_{j-1} passes z to the right. In this case,

$$\begin{array}{lll} SL(z) = \{1, \cdots, j-1\}, & SL'(z) = \{1, \cdots, i-1\}, & \\ & & i < j. \\ SR(z) = \{j, \cdots, n\}, & SR'(z) = \{i, \cdots, n+1\}, & \end{array}$$

Let M' be a tree partial to T', which consists of all the leaves $SL \cap SR'$ and the paths connecting them to the root of T. ($SM' = SL \cap SR' = \{i, \cdots, j-1\}$ is the set of indices of weights which have moved to the right of z.)

When w_{j-1} moves right it must stay left of $x'_j \leqq x_j$. $x'_j \neq x_j$ since we must make space for w_{j-1}. Consequently, w_j moves down left. x_j, the old position of w_j, is an internal node of T'; and as it is an ancestor of x'_{j-i}, it is an internal node of M' with all members of SM' belonging to its left subtree.

Construct a partial tree T^2 consisting of $L(z)$ and $R'(z)$ from which we have deleted the elements of SM'. This enables us to bring w_j up one level. Possibly w_j can be raised by more than that. However, for simplicity we shall raise it by exactly one level.

$$WT^2 = WL(z) + WR'(z) - w_j - WM' \geqq WT' = WL'(z) + WR'(z).$$

The inequality arises from the fact that T^2 and T' are built on the same weights ($ST' = ST^2 = \{1, \cdots, n+1\}$) and T' is optimal.

In case of equality, T^2 is an optimal tree in which no weight has moved right. Otherwise, $WT^2 > WT'$. Construct the tree T^3 from $L'(z)$ and $R(z)$. The elements of SM' are still missing. M' is a partial tree whose root is the root of T'. We may insert M' such that its internal nodes above x_j coincide with the corresponding nodes of $R(z)$. Node x_j of $R(z)$ can be moved down-right one level. This makes space for inserting the rest of M' (which includes all of its leaves).

$$WT^3 = WL'(z) + (WM' + w_j) + WR(z) < WL(z) + WR(z) = WT.$$

As T^3 and T have the same set of weights, this contradicts the optimality of T.

Case (c). w_j moves left of z. This case is analogous to (b), and the proof goes along the same lines. Q.E.D.

Theorem 1 is now an easy consequence of Lemma 6, since the optimal tree found in Lemma 6 fulfills all the requirements of Theorem 1.

Now instead of adding a weight let us consider deleting one. The following lemma will enable us to prove Theorem 2.

LEMMA 7. *Let $T[1, n, k]$ be an optimal tree. There exists an optimal tree $T'[2, n, k]$ such that $x'_i \leqq x_i$ for $i = 2, \cdots, n$.*

The proof is similar and, in some respects, simpler than that of Lemma 6 and, therefore, will not be pursued here.

We are now ready to prove Theorem 2. The first inequality of part (a), $b[i, j-1, k] \leqq b[i, j, k]$, is just Theorem 1. The second inequality, $b[i, j, k] \leqq b[i+1, j, k]$, follows from Lemma 7 just as Theorem 1 from Lemma 6.

Part (b), $b'[i, j-1, k] \leqq b'[i, j, k]$, is true since if $b'[i, j, k] < b'[i, j-1, k]$, then by the second inequality of part (a) and symmetry there exists an optimal tree $T'[i, j-1, k]$ such that

$$b'[i, j-1, k] \leqq b'[i, j, k] < b'[i, j-1, k].$$

This contradicts the definition of $T'[i, j-1, k]$ as an optimal tree with the smallest breakpoint. The second inequality follows similarly and concludes the proof of Theorem 2.

6. Applications and related problems.

The results of the previous sections lend themselves to several applications. First consider the K restricted Huffman problem [1]. Given n source words, each with a predetermined frequency of occurrence, construct a code with minimum average message length satisfying the restriction that no codeword has length greater than K. Just as in Huffman's original problem, we consider the order between the source words immaterial. The following lemma enables us to use the previous results to solve this problem.

LEMMA 8. *If $K \geqq \lceil \log_2 n \rceil$ and $w_1 \geqq \cdots \geqq w_n$, there exists an optimal K restricted tree for $(w_1, \cdots, w_n)$ such that the leaves associated with the weights preserve the order* (i.e., *x_i is left of x_{i+1}*, $i = 1, \cdots, n-1$). The proof is well known [3].

As a consequence of this lemma, we obtain an algorithm for the K-restricted Huffman problem:

(a) Sort the weights such that $w_1 \geqq w_2 \geqq \cdots \geqq w_n$.

(b) Apply the algorithm of § 4 to the new sequence. This is the algorithm which was proposed by Garey [1].

As previously mentioned, a nonrestricted version of the algorithm was first suggested by Knuth [4]. He used the algorithm to find an optimal alphabetic tree in which information may be stored in the internal nodes as well as in the leaves.

The extended algorithm may be applied to the K restricted version of Knuth's problem; i.e., find a tree of minimum cost in which weights are associated with all nodes, the order of the weights is preserved and no path has depth greater than K. The proof is essentially similar to that of § 5 and will not be presented.

Note that Knuth's original problem may be viewed as a special case of the K-restricted problem when K is sufficiently large ($K \geqq n$).

7. Alphabetic trees of degree $\sigma > 2$. Up to now, all algorithms heavily depended on the fact that we dealt with binary trees. Even though binary codes are of greatest importance, it is worthwhile to consider codes over an alphabet of more than two letters. Similarly, there are applications of σ-ary search trees for $\sigma > 2$.

Define a *tree of degree σ (σ-ary tree)* as consisting of a distinguished node, *root* and at most σ *subtrees* (the subtrees are ordered). Alphabetic σ-ary trees are defined similarly to alphabetic binary trees. Our object is to find optimal alphabetic σ-ary trees, i.e., such trees of minimum cost. We shall outline how to find optimal trees of nonrestricted depth. The extension to trees of restricted depth is straightforward.

Let an s-forest be a sequence of s trees. The cost of an s-forest is the sum of the costs of all its trees. Denote an optimal s-forest on weights $(w_i, \cdots, w_j)$ by $F_s[i, j]$. Note that $F_1[i, j] = T[i, j]$. The cost of the s-forest, $s > 1$, is

$$WF_s[i, j] = \min_{i \leqq b < j} \{WF_{s'}[i, b] + WF_{s-s'}[b+1, j]\}$$

for any s' $1 \leqq s' < s$. The cost of a σ-ary trees is

$$WT[i, j] = W_{ij} + WF_\sigma[i, j], \quad \text{where } W_{ij} = \sum_{r=i}^{j} w_r.$$

Using these equations we may extend the dynamic programming solutions of the previous sections to find optimal trees and optimal forests. When $\delta = j - i \geqq \sigma$, there is always an optimal σ-ary tree for $(w_i, \cdots, w_j)$ with exactly σ subtrees.

For each $\delta = 1, \cdots, n-1$, we shall find optimal σ-trees and s-forests for weights $(w_i, \cdots, w_j)$, $j = i + \delta$. The values of s for which we shall find optimal s-forests will depend on σ as follows: First, include the numbers $2, 4, 8, \cdots, 2^{\lfloor \log_2 \sigma \rfloor}$; next include the sequence $(s_0, \cdots, s_m)$, where $s_0 = \sigma$, $s_{i+1} = s_i - 2^{\lfloor \log_2 s_i \rfloor}$ and s_m is the last number which is not a power of 2. From the construction, it is clear that a forest of s_i trees can be constructed by combining a forest of s_{i+1} trees and a forest of $2^{\lfloor \log_2 s_i \rfloor}$ trees. As $m \leqq \lfloor \log_2 \sigma \rfloor$, we conclude that we need to consider at most $2\lfloor \log_2 \sigma \rfloor$ values of s.

Applying this extension to Gilbert and Moore's [2] algorithm yields an $n^3 \log \sigma$ algorithm.

Given a forest $F_s[i, j]$, let s be constructed as $s = s' + (s - s')$ in the above sequence; define the *breakpoint* as the index of the rightmost weight of the s'th tree.

Using this definition, we notice that Theorems 1, 2 hold also for s-forests and σ-ary trees. This enables us to take advantage of Knuth's observation and cut the running time down to $n^2 \log_2 \sigma$.

8. Alphabetic trees with letters of unequal cost. An interesting problem arises when we consider constructing an optimal alphabetic tree with letters of unequal cost. Specifically, for an alphabet of σ letters, let $(c_1, \cdots, c_\sigma)$ be the *cost of the letters.* The *cost of an edge* (a, b) in a σ-ary tree is c_i where b is the ith son of a. (In general, the σ-ary tree need not be full; i.e., not every node has σ sons. For $\sigma = 5$, a node may have a second and fourth son while lacking the first, third and fifth sons.) The *cost of a node* is the sum of the costs of all the edges of the unique path from the root to that node. For weights $(w_1, \cdots, w_n)$, an *alphabetic tree* is a σ-ary tree with weights on the nodes such that if $i<j$, then w_i is left of w_j. The *cost of an alphabetic tree* is $WT[1, n] = \sum_{i=1}^{n} p_i w_i$, where p_i is the cost of the node associated with w_i.

Our aim is to find, for given weights, an alphabetic tree of minimum cost (optimal tree). Note that, as the tree is alphabetic, we may not assume that the costs of the letters satisfy $c_1 \leqq c_2 \leqq \cdots \leqq c_\sigma$. (We may well have $c_1 < c_2$, $c_2 > c_3$.) We shall consider the case in which the weights are only at the leaves.

Let $T_{\alpha,\beta}[i, j]$ be an optimal tree for weights $(w_i, \cdots, w_j)$ in which the root has no son smaller than α and none greater than β. For every α,β,i,j $(1 \leqq \alpha \leqq \beta \leqq \sigma;\ 1 \leqq i \leqq j \leqq n)$, we shall find $WT_{\alpha,\beta}[i, j]$, the cost of an optimal tree $T_{\alpha,\beta}[i, j]$. (This implies that $T_{\alpha,\beta}$ has at least one edge).

The following equations will provide means to calculate $WT_{\alpha,\beta}[i, j]$:

$$WT_{\alpha,\alpha}[i, i] = c_\alpha w_i;$$

the tree $T_{\alpha,\alpha}[i, i]$ uses at least one edge, in this case that edge must be α.

$$WT_{\alpha,\beta}[i, i] = \min_{\alpha \leqq \gamma \leqq \beta} \{WT_{\gamma,\gamma}[i, i]\}, \qquad \alpha < \beta;$$

the root must have exactly one son (edge γ).

$$(2) \qquad WT_{\alpha,\alpha}[i, j] = c_\alpha W_{ij} + \min_{\substack{i \leqq b < j \\ 1 \leqq \gamma < \sigma}} \{WT_{\gamma,\gamma}[i, b] + WT_{\gamma+1,\sigma}[b+1, j]\},\ i<j.$$

The root has exactly one son (edge α). The son must have at least two sons; let the first be γ. We choose γ and b, the index of the rightmost weight of $T_{\gamma,\gamma}[i, b]$, so as to minimize the sum of the cost of the two trees—$WT_{\gamma,\gamma}[i, b] + WT_{\gamma+1,\sigma}[b+1, j]$.

$$WT_{\alpha,\beta}[i, j] = \min_{i \leqq b < j} \{WT_{\alpha+1,\beta}[i, j],\ WT_{\alpha,\alpha}[i, b] + WT_{\alpha+1,\beta}[b+1, j],\ WT_{\alpha,\alpha}[i, j]\},$$
$$\alpha < \beta,\ i < j.$$

We should consider three cases: In the first, it is not worthwhile to use edge α at all, hence the optimal tree is $WT_{\alpha+1,\beta}[i, j]$; in the second, α is truly the first edge, but there are additional edges. The cost of the optimal tree is $WT_{\alpha,\alpha}[i, b]$

$+ WT_{\alpha+1,\beta}[b+1, j]$. The last case is when only edge α is used (this might make some smaller edges available). The cost then is $WT_{\alpha,\alpha}[i, j]$. (Now we shall use (2) above, where $\gamma < \sigma$ and $b < j$, and hence we are not led into an infinite loop).

We compute $WT_{\alpha,\beta}[i, j]$ in order of increasing $j - i = p$, and for a fixed value of p in order of increasing $\beta - \alpha$. This insures that we never reference values which have not yet been computed.

This algorithm can be improved by use of the fact that adding an additional rightmost node does not cause any weight to move to the right. The statement and proof are similar to that of Theorems 1, 2. This enables us to reduce the computation time by a factor of n to $\sigma^2 n^2$.

Thus, if we consider σ fixed (e.g., $\sigma = 2$), the requirement for letters of different cost does not increase the known time complexity of the problem.

9. Alphabetic trees and nonalphabetic trees. The problem of alphabetic trees is closely related to that of nonalphabetic trees (i.e., trees in which the order of the weights is immaterial). However, the relation between the complexity of the two problems is unclear. For given weights, there are $n!$ times more nonalphabetic trees than alphabetic trees. This implies a larger domain for the nonalphabetic case. However, the lack of the order constraint may help us in our search.

We shall now give a closer look at some of the problems and show the relationship between the alphabetic and the nonalphabetic cases.

Case (a). Binary trees, weights at all nodes, letters of equal cost. Knuth [4] considered the alphabetic problem and found an n^2 algorithm. For the nonalphabetic case, there exists a linear solution as follows:

First notice that for weights $(w_1, \cdots, w_n)$ all optimal trees must satisfy

(i) if $w_i < w_j$, then $l_i \geqq l_j$;

(ii) all nodes, except perhaps those of the last two levels, have exactly two sons.

Without loss of generality, $n = 2^l - 1$ (otherwise add weights $(w_{n+1}, \cdots, w_{2^l-1} = 0)$, $2^l < 2n$, when the tree is found delete those weights). For such n, the optimal tree is the complete binary tree of depth $l - 1$. (The shape of the tree does not depend on the weights, only on their number.)

We now have to distribute the weights in the tree. This can be done in linear time by means of the median algorithm [5]. The deepest level will contain all nodes less than or equal to the median. The second deepest level will contain all nodes less than or equal the median of the remaining nodes.

$$\text{Time}\,(n) = CN + Cn/2 + Cn/4 + \cdots + C1$$

$$\leqq Cn(1 + \tfrac{1}{2} + \tfrac{1}{4} + \cdots) \leqq 2Cn.$$

C is the constant of the median algorithm. The internal order of the nodes at each level is arbitrary, since it does not change the cost of the tree.

Case (b). σ-ary trees, weights only at the leaves, letters of equal cost. The nonalphabetic case is Huffman's problem, for which his algorithm yields an $n \log n$ solution. In the alphabetic case with $\sigma = 2$, the T-C algorithm [6], [7] also yields an $n \log n$ algorithm. For $\sigma > 2$, we may apply the algorithm of § 7, the time bound of which is $n^2 \log \sigma$.

Case (c). Binary trees, weights at all nodes, letters of unequal cost. For the nonalphabetic case, as in (a) above, the shape of the tree is independent of the actual weights. Therefore, we can find a tree of n minimum cost vertices in time $n \log n$; number the nodes so that cost $(\mathrm{node}_i) \leqq$ cost (node_{i+1}), $i = 1, \cdots, n-1$.

Order the weights so that $w_1 \geqq w_2 \geqq \cdots \geqq w_n$; then associate with the ith node the ith weight. The optimality of the tree resulting from this algorithm is proven similarly to that of (a) above. The time bound on the algorithm is $n \log n$.

For alphabetic trees an algorithm similar to that of § 7 yields an n^2 solution.

Case (d). σ-ary trees, weights only at the leaves, letters of unequal cost. No polynomially bound algorithm is known—see [8] for an integer programming solution.

In § 8, we have given a $\sigma^2 n^2$ solution for the alphabetic case. It is not known whether it is possible to reduce the nonalphabetic case to the alphabetic case since no result similar to Lemma 8 of § 6 is known to hold.

Case (e). Restricting the maximum depth of the tree to K. For the alphabetic case, this causes an additional factor of K. For the nonalphabetic case, (a) and (c) retain their previous bounds, whereas the appropriate variant of (b) requires a reduction to the alphabetic case (see Lemma 8, § 6).

In conclusion, we may say that in most cases there are slightly better algorithms for the nonalphabetic case. However, there is a case (d) for which no polynomially bound algorithm is known for the nonalphabetic case, whereas the alphabetic case has a polynomial algorithm.

REFERENCES

[1] M. R. GAREY, *Optimal binary search trees with restricted maximal depth*, this Journal, 2 (1974), pp. 101–110.

[2] E. N. GILBERT AND E. F. MOORE, *Variable-length binary encodings*, Bell System Tech. J. 38 (1959), pp. 933–968.

[3] DONALD E. KNUTH, *The Art of Computer Programming, 1: Fundamental Algorithms*, Addison-Wesley, Reading, Mass., 1968.

[4] ———, *Optimum binary search trees*, Acta Informat., 1 (1971), pp. 14–25.

[5] ———, *The Art of Computer Programming, 3: Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.

[6] T. C. HU AND A. C. TUCKER, *Optimum computer search trees*, SIAM J. Appl. Math., 21 (1971), pp. 514–532.

[7] T. C. HU, *A new proof of the T-C algorithm*, Ibid., 25 (1973), pp. 83–94.

[8] R. M. KARP, *Minimum-redundancy coding for the discrete noiseless channel*, IEEE Trans. Information Theory, IT-7 (1961), pp. 27–39.