Acta Informatica 1, 14–25 (1971)
© by Springer-Verlag 1971

Optimum Binary Search Trees*

D. E. Knuth

Received June 22, 1970

One of the popular methods for retrieving information by its "name" is to store the names in a binary tree. To find if a given name is in the tree, we compare it to the name at the root, and four cases arise:

1. There is *no* root (the binary tree is empty): The given name is not in the tree, and the search terminates *unsuccessfully*.
2. The given name *matches* the name at the root: The search terminates *successfully*.
3. The given name is *less* than the name at the root: The search continues by examining the *left subtree* of the root in the same way.
4. The given name is *greater* than the name at the root: The search continues by examining the *right subtree* of the root in the same way.

Special cases of this method are the binary search and its variants (uncentered binary search; Fibonacci search) and the search-sort scheme of Wheeler-Berners Lee-Booth-Hibbard-Windley, *et al.* (see [1, 3, 7, 10]).

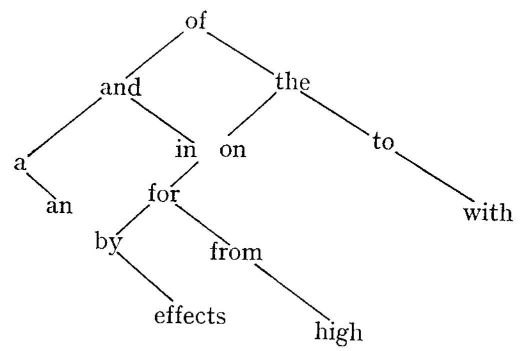
When all names in the tree are equally probable, it is not difficult to see that a best possible binary tree from the standpoint of average search time is one with minimum path length, namely the *complete binary tree* (see [9, pp. 400–401]). This is the tree which is implicitly present in one of the variants of the binary search method.

But when some names are known to be much more likely to occur than others, the best possible binary tree will not necessarily be balanced. For example, consider the following words and frequencies,

a	32
an	7
and	69
by	13
effects	6
for	15
from	10
high	8
in	64
of	142
on	22
the	79
to	18
with	9

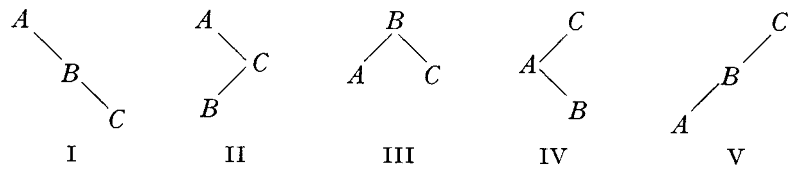
* The research reported here was supported by IBM Corporation.

showing words to be ignored in a certain KWIC indexing application [6, p. 124]. The best possible tree in this case turns out to be

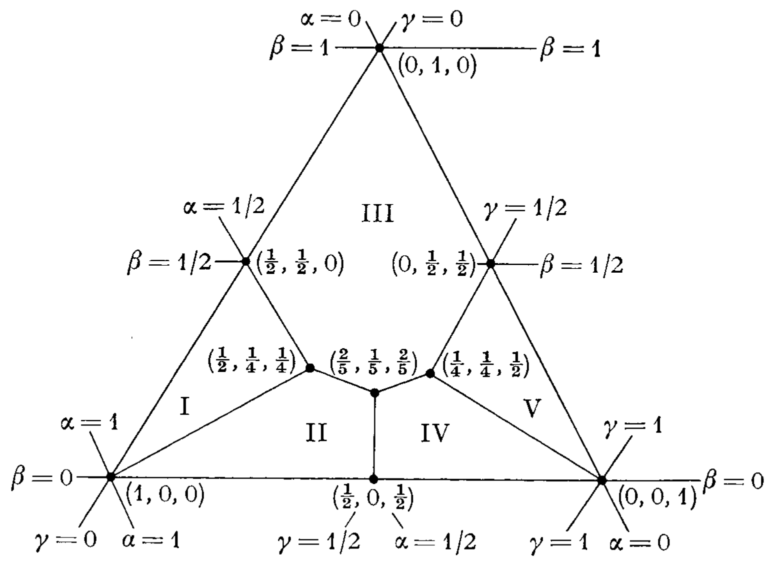


In this paper we discuss the question of finding such "optimal binary trees", when frequencies are given. The ordering property of the tree makes this problem more difficult than the standard "Huffman coding problem" (see [9, Section 2.3.4.5]).

For example, suppose that our words are A, B, C and the frequencies are α, β, γ. There are 5 binary trees with three nodes:



The following diagram shows the ranges of (α, β, γ) in which each of these trees is optimum, assuming that $\alpha+\beta+\gamma=1$:



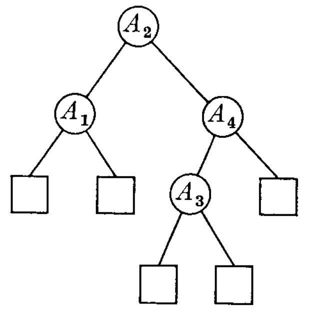
Note that it is sometimes best to put B at the root even when both A and C occur more frequently. And on the other hand, it is not sufficient simply to choose the root so as to equalize the left and right search probabilities as much as possible, contrary to a remark of Iverson [8, p. 144; 2, p. 318].

In general, there are $\binom{2n}{n}\frac{1}{n+1} \sim 4^n/n\sqrt{\pi n}$ binary trees with n nodes, so an exhaustive search for the optimum is out of the question. However, we shall show below that an elementary application of "dynamic programming," which is essentially the same idea used as the basis of the Cocke-Kasami-Younger-Earley parsing algorithm for context-free grammars [4], can be used to find an optimum binary search tree in order n^3 steps. By refining the method we will in fact cut the running time to order n^2.

In practice we want to generalize the problem, considering not only the frequencies with which a *successful* search is completed, but also the frequencies where *unsuccessful* searches occur. Thus we are given n names $A_1, A_2, \ldots, A_n$ and $2n+1$ frequencies $\alpha_0, \alpha_1, \ldots, \alpha_n; \beta_1, \beta_2, \ldots, \beta_n$. Here β_i is the frequency of encountering name A_i, and α_i is the frequency of encountering a name which lies between A_i and A_{i+1}; α_0 and α_n have obvious interpretations.

The key fact which makes this problem amenable to dynamic programming is that all subtrees of an optimum tree are optimum. If A_i appears at the root, then its left subtree is an optimum solution for frequencies $\alpha_0, \ldots, \alpha_{i-1}$ and $\beta_1, \ldots, \beta_{i-1}$; its right subtree is optimum for $\alpha_i, \ldots, \alpha_n$ and $\beta_{i+1}, \ldots, \beta_n$. Therefore we can build up optimum trees for all "frequency intervals" $\alpha_i, \ldots, \alpha_j$ and $\beta_{i+1}, \ldots, \beta_j$ when $i \leqq j$, starting from the smallest intervals and working toward the largest. Since there are only $(n+2)(n+1)/2$ choices of $0 \leqq i \leqq j \leqq n$, the total amount of computation is not excessive.

Consider the following binary tree:



(Square nodes denote empty or terminal positions where no names are stored.) The "weighted path length" P of a binary tree is the sum of frequencies times the level of the corresponding nodes; in the above example the score is

$$3\alpha_0 + 2\beta_1 + 3\alpha_1 + \beta_2 + 4\alpha_2 + 3\beta_3 + 4\alpha_3 + 2\beta_2 + 3\alpha_4.$$

In general, we can see that the weighted path length satisfies the equation

$$P = P_L + P_R + W,$$

where P_L and P_R are the weighted path lengths of the left and right subtrees, and $W = \alpha_0 + \alpha_1 + \cdots + \alpha_n + \beta_1 + \cdots + \beta_n$ is the "weight" of the tree, the sum of all frequencies. The weighted path length measures the relative amount of work needed to search the tree, when the α's and β's are chosen appropriately; therefore the problem of finding an optimum search tree is the problem of finding a binary tree of minimum weighted path length, with the weights applied from left to right in the tree.

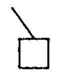
The above remarks lead immediately to a straightforward calculation procedure for determining an optimum search tree. Let P_{ij} and W_{ij} denote the weighted path length and the total weight of an optimum search tree for all words lying strictly between A_i and A_{j+1}, when $i \leqq j$; and let R_{ij} denote the index of the root of this tree, when $i < j$. The following formulas now determine the desired algorithm:

$$\begin{aligned} P_{ii} &= W_{ii} = \alpha_i, \quad \text{for} \quad 0 \leqq i \leqq n; \\ W_{ij} &= W_{i,j-1} + \beta_j + \alpha_j, \qquad\qquad (**) \\ P_{i,R_{ij}-1} + P_{R_{ij},j} &= \min_{i<k\leqq j}(P_{i,k-1} + P_{kj}) = P_{ij} - W_{ij}, \quad \text{for} \quad 0 \leqq i < j \leqq n. \end{aligned}$$

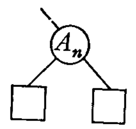
The problem of finding "best alphabetical encodings," considered by Gilbert and Moore in their classic paper [5], is easily seen to be a special case of the problem considered here, with $\beta_1 = \beta_2 = \cdots = \beta_n = 0$. Another closely related (but not identical) problem has been discussed by Wong [12]. In both cases the authors have suggested an algorithm for finding an optimum tree which is essentially identical to (**); Gilbert and Moore observe that the algorithm takes about $n^3/6$ iterations of the inner loop (choosing R_{ij} from among $j-i$ possibilities).

By studying the combinatorial properties of optimum binary trees more carefully, we can refine the algorithm somewhat.

Lemma. If $\alpha_n = \beta_n = 0$, an optimum binary tree may be obtained by replacing the rightmost terminal node



of the optimum tree for $\alpha_0, \ldots, \alpha_{n-1}$ and $\beta_0, \ldots, \beta_{n-1}$ by the subtree



Proof. By the formulas above, $W_{i,n} = W_{i,n-1}$ for $0 \leqq i < n$; $P_{nn} = \alpha_n = 0$; $R_{n-1,n} = n$; $P_{n-1,n} = 2\alpha_{n-1}$. We want to prove that $P_{in} = P_{i,n-1} + \alpha_{n-1}$ and $R_{in} = R_{i,n-1}$ for $0 \leqq i \leqq n-2$, and the proof is by induction on $n-i$. Consider the sums

$$P_{i,i} + P_{i+1,n}; \ldots; P_{i,n-2} + P_{n-1,n}; P_{i,n-1} + P_{n,n}.$$

By induction, these are respectively equal to

$$P_{i,i} + P_{i+1,n-1} + \alpha_{n-1}; \ldots; P_{i,n-2} + P_{n-1,n-1} + \alpha_{n-1}; P_{i,n-1}.$$

Let $R_{i,n-1}=r$; since

$$P_{i,n-1}=P_{i,r-1}+P_{r,n-1}+W_{i,n-1}\geqq P_{i,r-1}+P_{r,n-1}+\alpha_{n-1},$$

the minimum value in the above set of numbers is $P_{i,r-1}+P_{rn}$, hence we may take $R_{in}=r$.

Theorem. Adding a new name to the tree, which is greater than all other names, never forces the root of the optimum tree to move to the left. In other words, there is always a solution to the above equations such that

$$R_{0,n-1}\leqq R_{0,n},$$

when $n\geqq 2$.

Proof. We use induction on n, the result being vacuous when $n=1$. Since the optimum tree is a function of $\alpha_n+\beta_n$, we may assume that $\beta_n=0$. The method of proof is to start with $\alpha_n=0$; in this case the above lemma assures us of a matrix R_{ij} satisfying the desired condition. We will show that this condition can be maintained as α_n increases to arbitrarily high values.

Let α be a value such that the optimum tree is $\mathcal{T}$ when $\alpha_n=\alpha-\varepsilon$, but it is $\mathcal{T}'\neq\mathcal{T}$ when $\alpha_n=\alpha+\varepsilon$, for all sufficiently small $\varepsilon>0$. Assume further that the root of $\mathcal{T}'$ is less than (i.e., to the left of) the root of $\mathcal{T}$. The weighted path length of $\mathcal{T}$ is a linear expression of the form

$$l(\alpha_0)\,\alpha_0+l(\alpha_1)\,\alpha_1+\cdots+l(\alpha_n)\,\alpha_n+l(\beta_1)\,\beta_1+\cdots+l(\beta_n)\,\beta_n,$$

where $l(x)$ denotes the level associated with x; and the corresponding formula for $\mathcal{T}'$ is

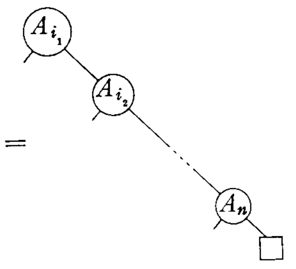
$$l'(\alpha_0)\,\alpha_0+l'(\alpha_1)\,\alpha_1+\cdots+l'(\alpha_n)\,\alpha_n+l'(\beta_1)\,\beta_1+\cdots+l'(\beta_n)\,\beta_n.$$

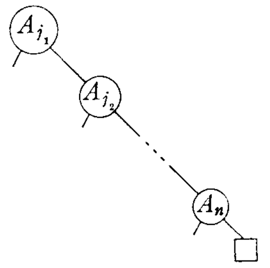
These two expressions become equal when $\alpha_n=\alpha$, and

$$l'(\alpha_n)<l(\alpha_n)$$

so that $\mathcal{T}'$ is better when $\alpha_n>\alpha$. When $\alpha_n=\alpha$, both trees are optimum.

Consider now the following diagrams:

$\mathcal{T}=$ 

$\mathcal{T}'=$ 

By our assumptions, $j_1<i_1$; $i_{l(\alpha_n)}=j_{l'(\alpha_n)}=n$. Since $j_1<i_1$, we can use induction and left-right symmetry of the theorem to conclude that $j_2\leqq i_2$. If $j_2<i_2$, similarly, we have $j_3\leqq i_3$. But since $l'(\alpha_n)<l(\alpha_n)$, $j_{l'(\alpha_n)}=n>i_{l'(\alpha_n)}$; hence $j_k=i_k$ for some k. Therefore we can replace the right subtree of A_{i_k} in $\mathcal{T}$ by the similar subtree in $\mathcal{T}'$, obtaining a binary tree $\mathcal{T}''$ whose weighted path length is equal to that of $\mathcal{T}'$ for all α_n. Since $\mathcal{T}''$ has the same root as $\mathcal{T}$, this argument shows that we need never move the root to the left as α increases.

Corollary. There is always a solution to conditions (**) above satisfying

$$R_{i,j-1}\leqq R_{i,j}\quad\text{and}\quad R_{i,j}\leqq R_{i+1,j},\quad\text{for}\quad 0\leqq i<j-1<n.$$

Proof. This is simply the result of the theorem applied to all subtrees, and using left-right symmetry.

The corollary suggests an algorithm which is much faster than the previous one, since we usually will not have to search the entire range $i<r\leqq j$ when determining R_{ij}. In fact, only $R_{i+1,j}-R_{i,j-1}+1$ cases need to be examined when R_{ij} is being calculated; summing for fixed $j-i$ gives a telescoping series which shows that the total amount of work is at worst proportional to n^2.

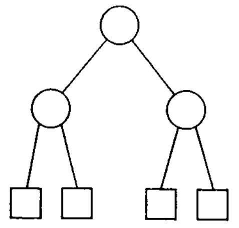
Summary, and Open Problems

The formulas above amount to a systematic method for finding optimum search trees, given the frequency of occurrence of each name in the tree as well as the frequencies of occurrence of names not in the tree. The number of steps is essentially proportional to the square of the number of names. An ALGOL program for the algorithm appears in the appendix, together with a detailed example from a compiler application.

Several open problems remain to be solved. Perhaps the most interesting is to obtain the best possible bound on the weighted path length in the optimum tree as a function of n, given arbitrary frequencies such that

$$\alpha_0+\alpha_1+\cdots+\alpha_n+\beta_1+\cdots+\beta_n=1.$$

For example, when $n=2$ the weighted path length is $\leqq 3$, and the worst case occurs when $\alpha_1=1$, $\alpha_0=\alpha_2=\beta_1=\beta_2=0$. The same bound applies when $n=3$, since the tree



obviously has weighted path length $\leqq 3$. It is not obvious what the best possible bounds are when $n>3$, although it is easy to see that the optimum weighted path length never exceeds $[\log_2(n+1)]+1$.

Another problem concerns the efficiency of the algorithm. Our n^2 algorithm essentially finds all of the optimal trees for $0\leqq i\leqq j\leqq n$. But if we discover by some means that $R_{0,n-1}\geqq 5$, it is unnecessary to determine $R_{i,n}$ for $1\leqq i\leqq 4$ when we compute $R_{0,n}$. There may be some way to arrange the calculation so that the method is less than order n^2 on the average.

A harder problem, but perhaps solvable, is to devise an algorithm which keeps its own frequency counts empirically, maintaining the tree in optimum form depending on the past history of the searches. Names that occur most frequently gradually move towards the root, etc. Perhaps some such updating method could be devised which would save more time than it consumes.

Another interesting problem is related to our first example. The optimum in the "of-and-the" case turned out to be obtainable by the following "top-down" rule: Place the most frequently occurring name at the root of the tree, then proceed similarly on the subtrees. Another plausible rule is to choose the root so as to equalize the total weight of the left and right subtrees as much as possible. Our example for $n=3$ shows that neither of these rules will produce an optimum tree in all cases, but it might be possible to give some quantitative estimate of how far from the optimum these methods can be.

The solution to any of these problems should provide further insight into the nature of optimum search trees.

I wish to thank Ronald L. Rivest for formulating a conjecture which led to the theorem in this paper, and John Bruno for correcting an error in my original proof of the lemma.

Appendix

The program below is written in ALGOL *W*, a refinement of ALGOL 60 due to Wirth and Hoare [11]. More than half of the code (the procedure *display*) is actually devoted to printing out the optimum tree in a reasonable pictorial fashion, one it has been found.

In order to try the algorithm on a fairly complicated test case, a count was made of all identifiers in about 25 example ALGOL *W* programs prepared by the author for an introductory programming course. The frequency of each reserved word was counted, as well as the frequency of occurrence of identifiers lying between adjacent reserved words. This led to the following data ($n=36$):

33			113		
	abs	1		**null**	8
5			2		
	and	6		**of**	5
0			0		
	array	9		**or**	5
26			30		
	begin	77		**procedure**	16
37			38		
	case	5		**real**	29
12			0		
	comment	95		**record**	2
54			0		
	div	12		**reference**	13
0			0		
	do	50		**rem**	9
23			0		
	else	16		**result**	0
0			23		
	end	77		**short**	0
15			11		
	false	2		**step**	5
0			0		
	for	35		**string**	5
36			99		
	go	1		**then**	34
0			2		
	goto	1		**to**	1
57			4		
	if	34		**true**	8
7			5		
	integer	37		**until**	34
142			4		
	logical	2		**value**	8
0			0		
	long	5		**while**	16
113			111		

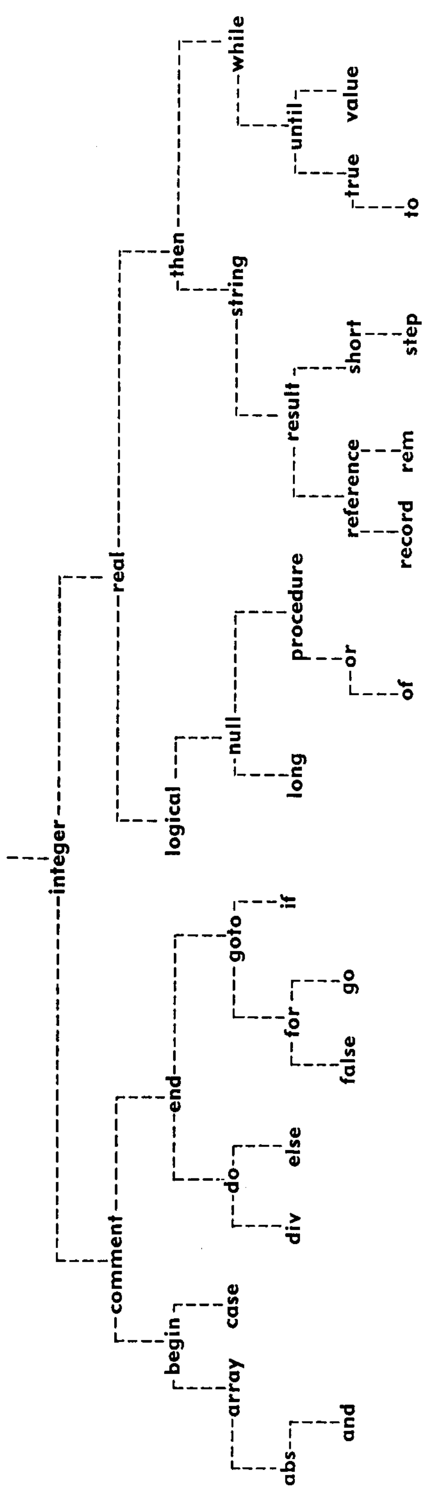
For example, any identifier starting with the letter *J*, *K*, or *L* would fall between integer and logical. The *R* matrix computed by the program is shown on the next page. The average search length for this fairly large tree came to less than 5.

The optimum tree printed out by the program appears below, as well as the quite different optimum tree obtained when the frequencies $\alpha_0, \alpha_1, \ldots, \alpha_{36}$ were set to zero. This shows that the "betweenness" frequencies can profoundly influence the nature of the optimum tree, so it is important to consider them.

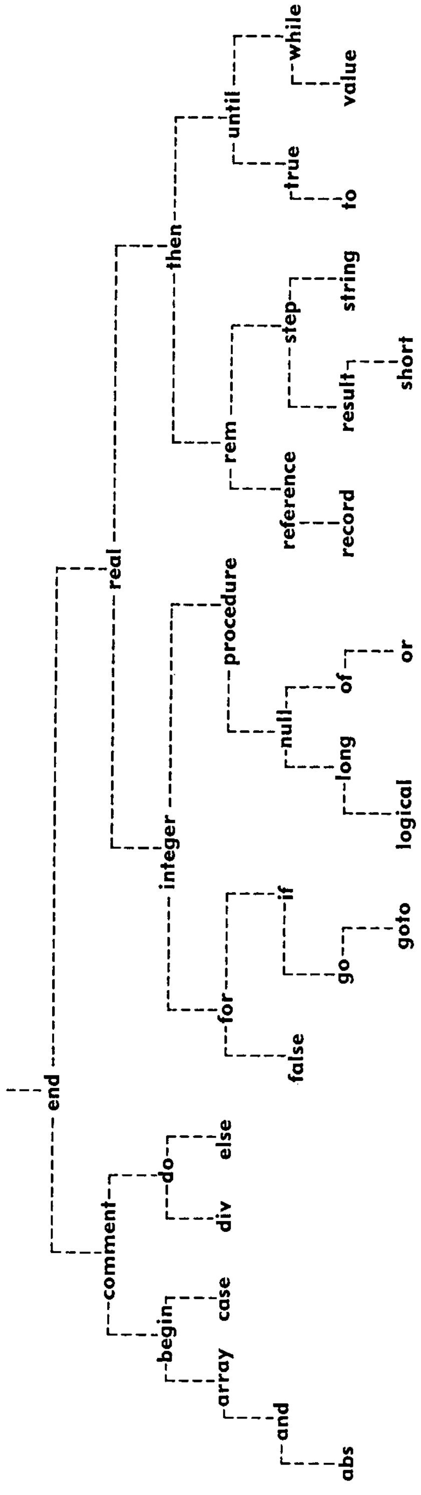
ALGOL *W* Program

```
begin comment Finding an 'optimum' search tree;
  string(10) array wd(1 :: 100); integer array a, b(0 :: 100);
  integer n;
  record node(string(10) info; integer col; reference(node) left, right);
  procedure display(integer value n; reference(node) value root);
    begin comment Draw a picture of binary tree referenced by 'root';
      reference(node) array active, waiting(1 :: n); string(132) line;
      integer k, newk; comment The number of nodes on the waiting list;
      reference(node) p;
      integer j; comment Counter used in colno procedure;
      procedure colno(reference(node) value r);
        begin comment Assign a column number to each node of the binary
            tree referenced by r;
          if r ≠ null then
            begin colno(left(r));
              col(r) := round(123*j/(n − 1)) + 4; j := j + 1;
              colno(right(r))
            end
        end colno;
      j := 0; colno(root);
      waiting(1) := root; k := 1;
      while k > 0 do
      begin line := " ";
        for j := 1 until k do
        begin comment Move waiting node to active area, and draw "|" lines
            down to it;
          active(j) := p := waiting(j);
          line(col(p)|1) := "|";
        end;
        write(line, line);
        newk := 0;
        for j := 1 until k do
        begin comment Put nodes descended from active nodes onto the
            waiting list, and prepare an appropriate line containing the 'info'
            of active nodes;
          integer cl, cr;
          p := active(j); cl := cr := col(p);
          if left(p) ≠ null then
            begin cl := col(left(p)); newk := newk + 1;
              waiting(newk) := left(p)
            end;
            if right(p) ≠ null then
              begin cr := col(right(p)); newk := newk + 1;
                waiting(newk) := right(p)
              end;
            for i := cl until cr do line(i|1) := "-";
            begin comment Center info(p) on line, about col(p);
              integer s; s := 0; while info(p) (s+1|1) ≠ " " do s := s+1;
              cl := col(p) - s div 2;
              for i := 0 until s do line(cl+i|1) := info(p) (i|1);
            end;
          end;
          write(line);
          k := newk
        end
      end display;
    n := 0; intfieldsize := 5;
    write("THE GIVEN FREQUENCIES ARE:");
    rloop: read(a(n), wd(n+1), b(n+1));
    write("                                        ", a(n));
    if wd(n+1) (0|1) ≠ "." then
    begin n := n+1;
      write("                                                    ", wd(n), b(n));
      go to rloop
    end;


    begin comment Find an n-node optimal tree, given relative frequency b(i) of
        encountering wd(i) and frequency a(i) of being between wd(i) and wd(i+1);
      integer array p, w, r(0::n, 0::n); comment p(i,j), w(i,j), r(i,j) denote
        respective the weighted path length, the total weight, and the root of the
        optimal tree for the words lying between wd(i) and wd(j+1), when i<j+1.
        The average search length in this tree is p(i,j)/w(i,j);
      reference(node) procedure createtree(integer value i, j);
        if i ≠ j then node(wd(r(i,j)), 0),
            createtree(i, r(i,j) - 1), createtree(r(i,j), j)) else null;
      for i := 0 until n do p(i,i) := w(i,i) := a(i);
      for i := 0 until n do for j := i+1 until n do
        w(i,j) := w(i,j-1) + b(j) + a(j);
      for k := 1 until n do for i := 0 until n-k do
      begin integer ik, mn, mx; ik := i+k;
        mx := if k=1 then ik else r(i, ik-1); mn := p(i, mx-1) + p(mx, ik);
        if k>1 then for j := mx+1 until r(i+1, ik) do
          if p(i, j-1) + p(j, ik) < mn then
            begin mn := p(i, j-1) + p(j, ik); mx := j end;
          p(i, ik) := mn + w(i, ik); r(i, ik) := mx
      end;
      write("AVERAGE PATH LENGTH IS", p(0,n)/w(0,n));
```



Optimum tree for the ALGOL-reserved-words application



Optimum tree when the α frequencies are ignored

0	1	1	3	4	4	4	6	6	6	6	6	6	6	6	6	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	16	16	16	16	16	16
0	0	2	3	4	4	6	6	6	6	6	6	6	6	6	6	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	16	16	16	16	16	16
0	0	0	3	4	4	6	6	6	5	6	6	6	6	6	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	16	16	16	16	16	16
0	0	0	0	4	4	6	6	6	6	6	6	6	6	6	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	16	16	16	16	16	16	16
0	0	0	0	0	5	6	6	6	6	8	8	8	8	10	10	10	10	10	10	10	10	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
0	0	0	0	0	0	6	6	6	6	8	8	8	8	10	10	10	10	10	10	10	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
0	0	0	0	0	0	0	7	8	8	8	8	10	10	10	10	10	10	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
0	0	0	0	0	0	0	0	8	8	10	10	10	10	10	10	14	14	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	19
0	0	0	0	0	0	0	0	0	9	10	10	10	10	12	12	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	17	17	19	19	19
0	0	0	0	0	0	0	0	0	0	10	10	10	10	12	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	17	17	19	19	19
0	0	0	0	0	0	0	0	0	0	0	11	12	12	14	14	16	16	16	16	16	16	16	17	17	17	17	17	17	17	17	17	17	17	19	19	19
0	0	0	0	0	0	0	0	0	0	0	0	12	12	14	14	16	16	16	16	16	16	17	17	17	17	17	17	17	17	17	17	17	17	19	19	23
0	0	0	0	0	0	0	0	0	0	0	0	0	13	14	15	16	16	16	16	16	16	17	17	17	17	17	17	17	17	17	17	17	17	19	19	23
0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	15	16	16	16	16	16	17	17	17	17	17	17	17	17	17	17	17	17	17	19	23	23
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	16	16	16	16	16	17	17	17	17	17	17	17	17	17	17	17	17	17	19	23	23
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	16	17	17	17	17	17	17	17	17	17	17	19	19	19	19	19	19	23	23	23
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	17	17	17	17	17	17	17	19	19	19	19	19	19	19	19	19	23	23	23
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	19	19	19	19	19	19	19	19	19	19	19	23	23	23	23	23	23	23
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	19	19	19	19	19	19	19	19	19	19	23	23	23	23	23	23	23
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	21	22	22	22	22	22	23	23	23	23	30	30	30	30	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21	22	22	22	22	23	23	23	23	23	30	30	30	30	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22	22	22	22	23	23	23	23	23	30	30	30	30	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	23	23	23	23	23	23	23	30	30	30	30	30	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	25	25	27	27	27	30	30	30	30	31	31	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	25	27	27	27	30	30	30	30	31	31	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26	27	28	28	30	30	30	31	31	31	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27	28	28	30	30	30	31	31	31	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28	28	30	30	30	31	31	31	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	29	30	31	31	31	31	31	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	31	31	31	31	31	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	31	31	31	31	31	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32	33	34	34	36
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33	34	34	36
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34	34	36
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35	36
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R matrix for the ALGOL-reserved-word application

```
    iocontrol(3);
    display(n, createtree(0, n));
    iocontrol(3); intfieldsize := 2;
    for i := 0 until n do
    begin iocontrol(2);
      for j := 0 until n do writeon(if i < j then r(i, j) else 0)
    end;
  end
end.
```

Note Added in Proof. T. C. Hu and A. C. Tucker have recently discovered a completely different way to find optimum binary search trees, in the special case that the β's are all zero. Their algorithm requires only $O(n)$ units of memory and $O(n \log n)$ units of time, when suitable data structures are employed.

References

1. Booth, A. D., Colin, A. J. T.: On the efficiency of a new method of dictionary construction. Information and Control **3**, 327–334 (1960).
2. Brooks, Frederick P., Jr., Iverson, Kenneth E.: Automatic data processing, System/360 edition. Wiley 1969.
3. Douglas, A. S.: Techniques for the recording of, and reference to data in a computer. The Computer Journal **2**, 1–9 (1959).
4. Earley, Jay: An efficient context-free parsing algorithm. Communications of the ACM **13**, 94–102 (1970).
5. Gilbert, E. N., Moore, E. F.: Variable-length binary encodings. The Bell System Technical Journal **38**, 933–968 (1959).
6. Helbich, Jan: Direct selection of keywords for the KWIC index. Information Storage and Retrieval **5**, 123–128 (1969).
7. Hibbard, Thomas N.: Some combinatorial properties of certain trees with applications to searching and sorting. Journal of the ACM **9**, 13–28 (1962).
8. Iverson, Kenneth E.: A programming language. Wiley 1962.
9. Knuth, Donald E.: The art of computer programming, **1**: Fundamental algorithms. Addison-Wesley 1968.
10. Windley, P. F.: Trees, forests and rearranging. The Computer Journal **3**, 84–88, 174, 184 (1960).
11. Wirth, Niklaus, Hoare, C. A. R.: A contribution to the development of ALGOL. Communications of the ACM **9**, 413–431 (1966).
12. Wong, Eugene: A linear search problem. SIAM Review **6**, 168–174 (1964).

Prof. Dr. Donald E. Knuth
Stanford University
Computer Science Department
Stanford, Calif. 94305
U.S.A.