SIAM J. COMPUT.
Vol. 6, No. 2, June 1977

A BEST POSSIBLE BOUND FOR THE WEIGHTED PATH LENGTH OF BINARY SEARCH TREES*

KURT MEHLHORN†

Abstract. The weighted path length of optimum binary search trees is bounded above by $\sum \beta_i + 2\sum \alpha_j + H$ where H is the entropy of the frequency distribution, $\sum \beta_i$ is the total weight of the internal nodes, and $\sum \alpha_j$ is the total weight of the leaves. This bound is best possible. A linear time algorithm for constructing nearly optimal trees is described.

Key words. binary search tree, complexity, average search time, entropy

One of the popular methods for retrieving information by its "name" is to store the names in a binary tree. We are given n names $B_1, B_2, \cdots, B_n$ and $2n+1$ frequencies $\beta_1, \cdots, \beta_n, \alpha_0, \cdots, \alpha_n$ with $\sum \beta_i + \sum \alpha_j = 1$. Here β_i is the frequency of encountering name B_i, and α_j is the frequency of encountering a name which lies between B_j and B_{j+1}, α_0 and α_n have obvious interpretations [4].

A binary search tree T for the names $B_1, B_2, \cdots, B_n$ is a tree with n interior nodes (nodes having two sons), which we denote by circles, and $n+1$ leaves, which we denote by squares. The interior nodes are labeled with the B_i in increasing order from left to right and the leaves are labeled with the intervals (B_j, B_{j+1}) in increasing order from left to right. Let b_i be the distance of interior node B_i from the root and let a_j be the distance of leaf (B_j, B_{j+1}) from the root. To retrieve a name X, b_i+1 comparisons are needed if $X = B_i$ and a_j comparisons are required if $B_j < X < B_{j+1}$. Therefore we define the weighted path length of tree T as:

$$P = \sum_{i=1}^{n} \beta_i(b_i+1) + \sum_{j=0}^{n} \alpha_j a_j.$$

It is equal to the expected number of comparisons needed to retrieve a name.

In [4] D. E. Knuth gives an algorithm for constructing an optimum binary search tree, i.e., a tree with minimal weighted path length. His algorithm operates in $O(n^2)$ units of time and $O(n^2)$ units of space. In [6] we discuss the following "rule of thumb" for constructing nearly optimal binary search trees: choose the root so as to equalize the total weight of the left and right subtree as much as possible, then proceed recursively. The weighted path length of a tree constructed according to this rule is bounded above by $2+1.44 \cdot H$, where $H = \sum \beta_i \log(1/\beta_i) + \sum \alpha_j \log(1/\alpha_j)$ is the entropy of the frequency distribution. This bound was recently improved by P. J. Bayer [1] to $2+H$. Here we discuss a different rule of thumb suggested by [3] and prove the upper bound $1+\sum \alpha_j + H$ for the weighted path length. This bound is best possible.

The rule presented here as well as the rules described in [6] can be implemented to work in linear time and space ([2]).

* Received by the editors September 24, 1975, and in revised form June 1, 1976.

† Universität des Saarlandes, 66 Saarbrücken, West Germany

We describe and analyze an approximation algorithm. The algorithm constructs binary search trees in a top-down fashion. It uses bisection on the set

$$\left\{s_i;\, s_i=\sum_{p=0}^{i-1}(\alpha_p+\beta_p)+\beta_i+\frac{\alpha_i}{2}\quad\text{and}\quad 0\leqq i\leqq n\right\},$$

i.e., the root (k) is determined such that $s_{k-1}\leqq\frac{1}{2}$ and $s_k\geqq\frac{1}{2}$. It then proceeds recursively on the subsets $\{s_i;\, i\leqq k-1\}$ and $\{s_i;\, i\geqq k\}$. In the definition of the s_i's we assumed $\beta_0=0$ for ease of writing. The main program

begin
  let $s_i\leftarrow\sum_{p=0}^{i-1}(\alpha_p+\beta_p)+\beta_i+\alpha_i/2$ for $0\leqq i\leqq n$;
  construct-tree (0, n, 0, 1)

end

uses the recursive procedure construct-tree:

procedure construct-tree (i, j, cut, l);
comment we assume that the actual parameters of any call of construct-tree satisfy the following conditions.
(1) i and j are integers with $0\leqq i<j\leqq n$,
(2) l is an integer with $l\geqq 1$,
(3) $cut=\sum_{p=1}^{l-1}x_p2^{-P}$ with $x_p\in\{0, 1\}$ for all p,
(4) $cut\leqq s_i\leqq s_j\leqq cut+2^{-l+1}$.
A call construct-tree (i, j, —, —,) will construct a binary search tree for the nodes (i+1), · · · ,(j) and the leaves [i], · · · ,[j];
begin
if $i+1=j$ (Case A)
then return the tree shown in Fig. 1.
else comment we determine the root so as to bisect the interval
  (cut, $cut+2^{-l+1}$)
  begin
  determine k such that
  (5) $i<k\leqq j$
  (6) $k=i+1$ or $s_{k-1}\leqq cut+2^{-l}$
  (7) $k=j$ or $s_k\geqq cut+2^{-l}$
  comment k exists because the actual parameters are supposed to satisfy condition (4);
  if $k=i+1$ (Case B)
  then return the tree shown in Fig. 2;
  if $k=j$ (Case C)
  then return the tree shown in Fig. 3;
  if $i+1<k<j$ (Case D)
  then return the tree shown in Fig. 4;
  end
end procedure construct-tree;

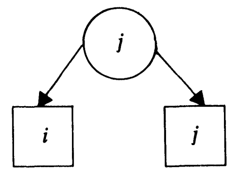

FIG. 1

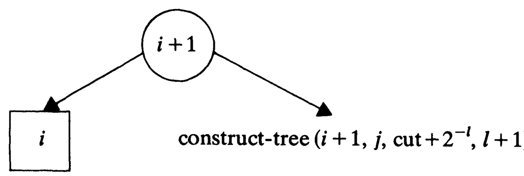

FIG. 2

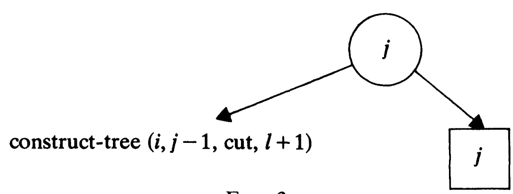

FIG. 3

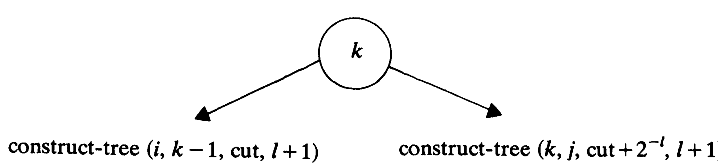

FIG. 4

LEMMA. *The approximation algorithm constructs a binary search tree whose weighted path length* P_{approx} *is bounded above by* $1+\sum \alpha_j + H$.

Proof. We state several simple facts.

FACT 1. *If the actual parameters of a call* construct-tree (i, j, cut, l) *satisfy conditions* (1) *to* (4) *and* $i+1 \neq j$, *then a* k *satisfying conditions* (5) *to* (7) *exists and the actual parameters of the recursive calls of* construct-tree *initiated by this call again satisfy conditions* (1) *to* (4).

Proof. Assume that the parameters satisfy conditions (1) to (4) and that $i+1 \neq j$. In particular, $cut \leqq s_j \leqq cut + 2^{-l+1}$. Suppose, that there is no k, $i < k \leqq j$, with $s_{k-1} \leqq cut + 2^{-l}$ and $s_k \geqq cut + 2^{-l}$. Then either for all k, $i < k \leqq j$, $s_k < cut + 2^{-l}$ or for all k, $i < k \leqq j$, $s_k > cut + 2^{-l}$. In the first case $k = j$ satisfies (6) and (7), in the second case $k=i+1$ satisfies (6) and (7). This shows that k always exists. It remains to show that the parameters of the recursive calls satisfy again (1) and (4). This follows immediately from the fact that k satisfies (5) to (7) and that $i+1 \neq j$ and hence $s_k \geqq cut + 2^{-l}$ in Case B and $s_{k-1} \leqq cut + 2^{-l}$ in Case C. Q.E.D.

FACT 2. *The actual parameters of every call of* construct-tree *satisfy conditions* (1) *to* (4) (*if the arguments of the top-level call do*).

Proof. The proof is by induction, Fact 1 and the observation that the actual parameters of the top-level call construct-tree $(0, n, 0, 1)$ satisfy (1) to (4). Q.E.D.

We say that node Ⓗ (leaf $\boxed{h}$ resp.) is constructed by the call construct-tree (i, j, cut, l) if $h=j$ ($h=i$ or $h=j$) and Case A is taken or if $h=i+1$ ($h=i$) and Case B is taken or if $h=j$ ($h=j$) and Case C is taken or if $h=k$ and Case D is taken. Let b_i be the depth of node ⓘ and let a_j be the depth of leaf $\boxed{j}$ in the tree returned by the call construct-tree $(0, n, 0, 1)$.

FACT 3. *If node* Ⓗ (*leaf* $\boxed{h}$ *is constructed by the call* construct-tree (i, j, cut, l), *then* $b_h + 1 = l$ ($a_h = l$).

Proof. The proof is by induction on l.

FACT 4. *If node* Ⓗ (*leaf* $\boxed{h}$) *is constructed by the call* construct-tree (i, j, cut, l), *then* $\beta_h \leqq 2^{-l+1}$ ($\alpha_h \leqq 2^{-l+2}$).

Proof. The actual parameters of the call satisfy condition (4) by Fact 2. Thus

$$2^{-l+1} \geqq s_j - s_i = (\alpha_i + \alpha_j)/2 + \beta_{i+1} + \alpha_{i+1} + \cdots + \beta_j$$
$$\geqq \beta_h \text{ (resp. } \alpha_h/s). \qquad \text{Q.E.D.}$$

FACT 5. *The weighted path length* P_{approx} *of the tree constructed by the approximation algorithm is bounded above by* $\sum \beta_j + 2\sum \alpha_j + H$.

Proof.

$$P_{\text{approx}} = \sum \beta_i(b_i + 1) + \sum \alpha_j a_j$$
$$\leqq \sum \beta_i(\log(1/\beta_i) + 1) + \sum \alpha_j(\log(1/\alpha_j) + 2)$$
$$\leqq \sum \beta_j + 2 \cdot \sum \alpha_j + H. \qquad \text{Q.E.D.}$$

THEOREM. *Let* $\alpha_0, \beta_1, \alpha_1, \cdots, \beta_n, \alpha_n$ *be any frequency distribution, let* P_{opt} *be the weighted path length of the optimum binary search tree for this distribution, let* P_{approx} *be the weighted path length of the tree constructed by the approximation algorithm, and let* $H = -\sum \beta_i \log \beta_i - \sum \alpha_j \log \alpha_j$ *be the entropy of the frequency distribution. Then*

$$P_{\text{opt}} \leqq P_{\text{approx}} \leqq \sum \beta_j + 2 \cdot \sum \alpha_j + H.$$

Furthermore, this upper bound is the best possible in the following sense: if $c_1 \sum \beta_i + c_2 \sum \alpha_j + c_3 \cdot H$ *is an upper bound for* P_{opt}, *then* $c_1 \geqq 1$, $c_2 \geqq 2$, *and* $c_3 \geqq 1$.

Proof. The first part of the theorem follows from the preceding lemma. The second part is proven by exhibiting suitable frequency distributions:

$c_1 \geqq 1$: Take $n=1$, $\alpha_0 = \alpha_1 = 0$ and $\beta_1 = 1$.
$c_2 \geqq 2$: Take $n=2$, $\alpha_0 = \alpha_2 = \beta_1 = \beta_2 = 0$, $\alpha_1 = 1$.
$c_3 \geqq 1$: Take $n = 2^k - 1$, $\beta_1 = 0$ for all i and $\alpha_j = 2^{-k}$ for all j.

It is easy to see that the complete binary tree is the optimal binary search tree for this distribution. Thus

$$H = \log(n+1) = k = \sum_{\text{leaves}} (1/2^k) \cdot k = P_{\text{opt}}. \qquad \text{Q.E.D.}$$

REFERENCES

[1] P. J. BAYER, *Improved bounds on the cost of optimal and balanced binary search trees*, M.Sc. thesis, Mass. Inst. of Tech., Cambridge, MA, 1975.
[2] M. L. FREDMAN, *Two applications of a probabilistic search technique: Sorting X + Y and building balanced search trees*, 7th Symp. on Theory of Computing, Albuquerque, NM., 1975.
[3] E. N. GILBERT AND E. F. MOORE, *Variable-length binary encodings*, Bell System Tech. J., 38 (1959), pp. 933–968.
[4] D. E. KNUTH, *Optimum binary search trees*, Acta Informatica, 1 (1971), pp. 14–25.
[5] ———, *The Art of Computer Programming*, vol. 3, Addison-Wesley, Reading, MA., 1973.
[6] K. MEHLHORN, *Nearly optimal binary search trees*, Acta Informatica, 5 (1975), pp. 287–295.