

Nearly Optimal Binary Search Trees

Kurt Mehlhorn

Received January 8, 1975

Summary. We discuss two simple strategies for constructing binary search trees: "Place the most frequently occurring name at the root of the tree, then proceed similarly on the subtrees" and "choose the root so as to equalize the total weight of the left and right subtrees as much as possible, then proceed similarly on the subtrees." While the former rule may yield extremely inefficient search trees, the latter rule always produces nearly optimal trees.

"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, \dots, B_n and $2n+1$ frequencies $\beta_1, \dots, \beta_n, \alpha_0, \dots, \alpha_n$ 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]. We may always assume w.l.o.g. that $\beta_i + \alpha_i + \beta_{i+1} \neq 0$ for all i . Otherwise, the i -th (or the $(i+1)$ -th) key might as well be removed.

A binary search tree T 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 labelled by the B_i in increasing order from left to right and the leaves are labelled by 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_i 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_i comparisons are required if $B_j < X < B_{j+1}$. Therefore we define the weighted path length of tree T as:

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

D. E. Knuth [4] gives an algorithm for constructing an optimum binary search tree, i.e. a tree with minimal weighted path length. His algorithm has $O(n^2)$ time complexity and $O(n^2)$ space complexity. This is prohibitive for most applications. He also describes two rules of thumb for constructing binary search trees, both of which can be implemented in $O(n \log n)$ time and $O(n)$ space (in the appendix we sketch such an implementation). Recently M. Fredman [1] improved on this and exhibited an algorithm which works in $O(n)$ units of time and space.

Rule I: Place the most frequently occurring name at the root of the tree, then proceed similarly on the subtrees.

Rule II: Choose the root so as to equalize the total weight of the left and right subtrees as much as possible, then proceed similarly on the subtrees.

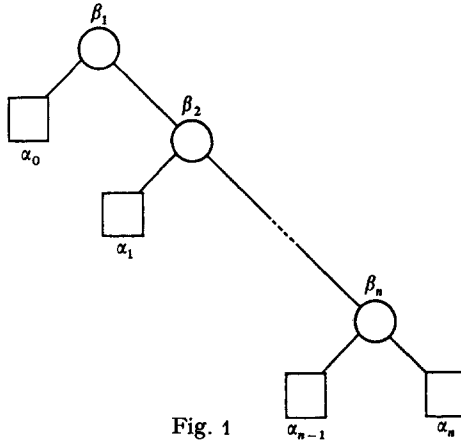


Fig. 1

Knuth poses the problem to give some quantitative estimate of how far from the optimum these methods can be. We answer this question. While Rule I may produce “bad” search trees, Rule II always yields nearly optimal trees. This fact was empirically known for some time.

We discuss first Rule I:

Place the most frequently occurring name at the root, then proceed similarly on the subtrees. A simple example shows that this rule yields “bad” search trees.

Let

$$n = 2^k - 1, \quad \beta_i = 2^{-k} + \varepsilon_i \quad \text{with} \quad \sum_{i=1}^n \varepsilon_i = 2^{-k}$$

and

$$\varepsilon_1 > \varepsilon_2 > \dots > \varepsilon_n > 0 \quad \text{for} \quad 1 \leq i \leq n \quad \text{and} \quad \alpha_j = 0 \quad \text{for} \quad 0 \leq j \leq n.$$

Rule I yields the tree of Fig. 1. Its weighted path length is:

$$\begin{aligned} P_1 &= \sum_{i=1}^n \beta_i (b_i + 1) + \sum_{j=0}^n \alpha_j a_j \\ &= \sum_{i=1}^n \beta_i i \\ &\geq 2^{-k} \sum_{i=1}^n i = 2^{-k} \frac{n(n+1)}{2} \geq \frac{n}{2}. \end{aligned}$$

Fig. 2 shows a balanced tree for the same frequency distribution. Its weighted path length is bounded by:

$$\begin{aligned} P_2 &= \sum_{i=1}^n \beta_i (b_i + 1) + \sum_{j=0}^n \alpha_j a_j \\ &\leq 2^{-(k-1)} \sum_{i=1}^n (b_i + 1) \\ &\leq 2^{-(k-1)} \sum_{l=1}^k 2^{(l-1)} \cdot l \leq 2 \cdot \log n. \end{aligned}$$

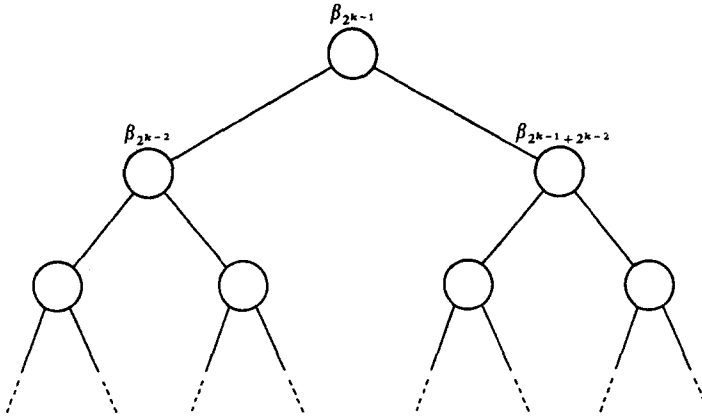


Fig. 2

Thus $P_1/P_2 > 4^{-1} n/\log n$. This example is a strong argument against the usefulness of Rule I.

We turn now to Rule II:

Choose the root so as to equalize the total weight of the left and right subtree as much as possible. Ties are broken arbitrarily.

W. A. Walker and C. C. Gotlieb [7] did empirical studies of a modification of this rule and report that it produces nearly optimal search trees. In the following we will prove that Rule II yields fairly good trees in all cases. The key idea of our proof is to show that the weights of the subtrees along any path essentially form a geometrically decreasing sequence. The weight of a tree is the sum of the probabilities of all nodes and leaves of this tree.

We need to define a few constants. Let

$$\varepsilon = \frac{1}{2}\sqrt{5} - 1 \text{ and } \delta = \frac{1}{2} + \varepsilon = \frac{1}{2}(\sqrt{5} + 1). \text{ Then } \delta^2 = \frac{1}{2} - \varepsilon.$$

$1/\delta$ is the golden ratio.

Lemma 1. Let T be a binary tree which is constructed according to Rule II. Let B be an interior node with distance 2 from the root. Let w_0 be the total weight of T , w_1 be the total weight of the direct subtree of T , which contains B , and let w_2 be the total weight of the tree with root B . Then either

$$w_1 \leq \delta w_0 \text{ or } w_2 \leq \delta^2 w_0.$$

Proof. We may assume w.l.o.g. that B lies in the left subtree of T and that the total weight w_0 of T is 1. If the weight w_1 of the left subtree of T is not greater than δ then we are done. Otherwise, consider Fig. 3. Let e be the weight of the right subtree of T , d be the weight of the root, c be the weight of the rightmost leaf in the left subtree of T , b be the weight of the rightmost interior node in the left subtree of T , and a be the weight of the remainder of the left subtree of T . By assumption

$$w_1 = a + b + c = 1/2 + \gamma$$

for some $\gamma > \varepsilon$. Then $d + e = 1/2 - \gamma$.

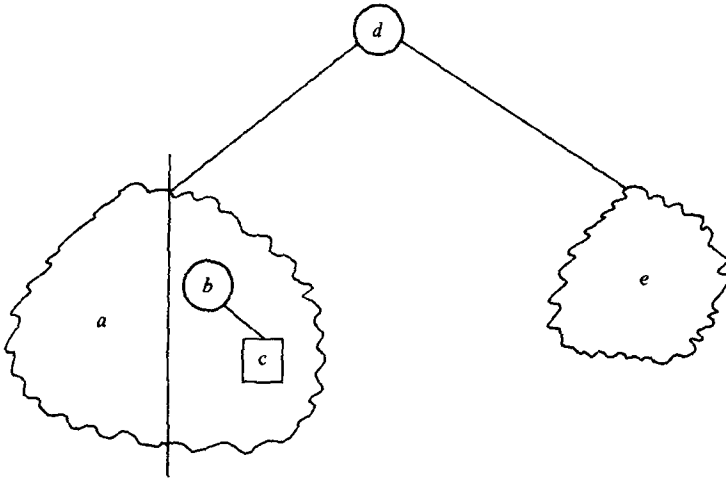


Fig. 3. The tree T

Rule II did not choose the node with weight b as the root of T . If the node with weight b were chosen as the root of T then the difference between the weights of the left and right subtree would be $|a - (c + d + e)|$. Since the node with weight b was not taken as the root

$$|a - (c + d + e)| \geq a + b + c - e. \tag{*}$$

If $a - (c + d + e)$ is positive then

$$a - (c + d + e) \geq a + b + c - e$$

and hence

$$0 \geq b + 2c + d.$$

Thus

$$0 \geq b + c + d.$$

This contradicts our assumption $\beta_i + \alpha_i + \beta_{i+1} \neq 0$ for all i . Therefore $a - (c + d + e)$ is negative.

In this case we infer from (*)

$$c + d + e - a \geq a + b + c - e$$

which yields

$$d + 2e \geq b + 2a.$$

Thus

$$1 - 2\gamma \geq d + 2e \geq b + 2a = \frac{1}{2} + \gamma + a - c.$$

Hence

$$a - c \leq 1/2 - 3\gamma, \quad a + c \leq 1/2 + \gamma,$$

so

$$a \leq 1/2 - \gamma < 1/2 - \epsilon.$$

If B is the root of the left subtree of the left subtree then the entire tree with root B is part of the structure with weight a and hence $w_2 \leq a < 1/2 - \epsilon$. Otherwise, we have the following picture of weights (see Fig. 4) with $a = x + y + z$ and $w_2 = z + b + c$.

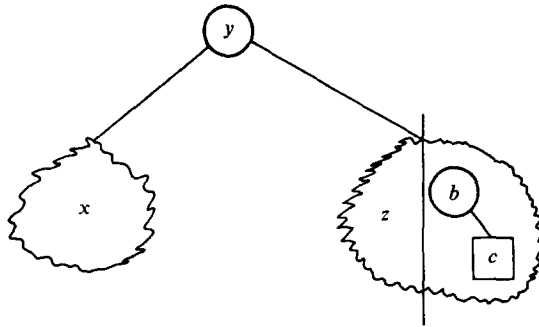


Fig. 4. The left subtree of tree T

Assume $w_2 > 1/2 - \epsilon$. Since b was not at level 1, we must have

$$z + b + c - x \leq a - c$$

hence

$$1/2 - \epsilon < w_2 = z + b + c \leq a + x - c.$$

Furthermore

$$\begin{aligned} 1 - 2\epsilon &< z + b + c + a + x - c \\ &= z + x + a + b \\ &\leq 2a + b \\ &\leq 1 - 2\gamma < 1 - 2\epsilon, \end{aligned}$$

a contradiction.

q.e.d.

Lemma 2. Let T be a binary tree which is constructed according to Rule II and let B be an interior node with distance b from the root. Let w be the total weight of the subtree with root B . Then

$$w \leq \delta^{(b-1)}.$$

Proof. The claim is obvious for $b \leq 1$. Otherwise, let $B_{k_0}, B_{k_1}, \dots, B_{k_b} = B$ be the nodes on the path from the root to B and let w_i be the weight of the subtree with root B_{k_i} . We show: for all i either

$$w_{i-1} \leq \delta^{i-1}$$

or

$$w_i \leq \delta^i.$$

For $i = 2$ this follows from Lemma 1. If $i > 2$ then either $w_{i-2} \leq \delta^{i-2}$ or $w_{i-1} \leq \delta^{i-1}$ by induction hypothesis. In the second case, we are done, in the first case, we apply Lemma 1 and obtain: either

$$w_{i-1} \leq \delta \cdot w_{i-2} \leq \delta^{i-1}$$

or

$$w_i \leq \delta^2 w_{i-2} \leq \delta^i.$$

Hence

$$\begin{aligned} w = w_b &\leq \min(w_b, w_{b-1}) \\ &\leq \delta^{(b-1)}. \end{aligned}$$

q.e.d.

Given any frequency distribution $\alpha_0, \beta_1, \dots, \beta_n, \alpha_n$, let T be the tree which is constructed according to rule II. Let b_i be the distance of interior node B_i from the root, let α_j be the distance of leaf (B_j, B_{j+1}) from the root, let B_{i_j} be the father of leaf (B_j, B_{j+1}) , and let w_i be the total weight of the subtree with root B_i .

Lemma 2 implies

$$\beta_i \leq w_i \leq \delta^{(b_i-1)}$$

and

$$\alpha_j \leq w_{i_j} \leq \delta^{(a_j-2)}.$$

Thus

$$b_i + 1 \leq (\log 1/\delta)^{-1} \cdot \log 1/\beta_i + 2$$

and

$$\alpha_j \leq (\log 1/\delta)^{-1} \cdot \log 1/\alpha_j + 2.$$

This gives the following upper bound for the weighted path length P_{balanced} of a tree constructed according to Rule II.

$$\begin{aligned} P_{\text{balanced}}(\alpha_0, \beta_1, \dots, \beta_n, \alpha_n) &= \sum_{i=1}^n \beta_i (b_i + 1) + \sum_{j=0}^n \alpha_j a_j \\ &\leq \sum_{i=1}^n \beta_i (2 + c \cdot \log 1/\beta_i) + \sum_{i=0}^n \alpha_i (2 + c \log 1/\alpha_i) \\ &\leq 2 + c \cdot (\sum \beta_i \log 1/\beta_i + \sum \alpha_j \log 1/\alpha_j) \end{aligned}$$

with $c = (\log 1/\delta)^{-1} = (1 - \log(\sqrt{5} - 1))^{-1}$.

Theorem 1. Given any frequency distribution $\alpha_0, \beta_1, \dots, \beta_n, \alpha_n$ with $\sum \beta_i + \sum \alpha_i = 1$, Rule II yields a tree whose weighted path length P_{balanced} is bounded above by

$$2 + (1 - \log(\sqrt{5} - 1))^{-1} \cdot H$$

where $H = \sum \beta_i \log 1/\beta_i + \sum \alpha_j \log 1/\alpha_j$ is the entropy of the frequency distribution.

We do not know if Theorem 1 is best possible. However, whenever $c_1 + c_2 \cdot H$ is an upper bound on the weighted path length of balanced trees, then $c_1 \geq 2$; e.g. let $\alpha_0 = \epsilon, \alpha_1 = 1 - 2\epsilon, \alpha_2 = \epsilon, \beta_1 = \beta_2 = 0$ for some small $\epsilon > 0$. It is also easy to show that $c_2 \geq 1$ is necessary. Rissanen [6] shows that $c_2 = 1$ is also sufficient in the special case that the weight is concentrated in the leaves ($\beta_i = 0$ for all i); $H + 3$ is an upper bound for P_{balanced} in this case. Our Theorem 1 yields $c_2 = (1 - \log(\sqrt{5} - 1))^{-1} \approx 1.44$ for the general case.

In the case of optimum binary search trees $H + 3$ is an upper bound on the weighted path length [5]. A least upper bound on P_{opt} in terms of n , the number of names, was given by Hu and Tan [2].

We proceed now to prove a lower bound on the weighted path length P_{opt} of any optimal binary search tree.

Theorem 2. Let $\alpha_0, \beta_1, \dots, \beta_n, \alpha_n$ be any frequency distribution with $\sum \beta_i + \sum \alpha_i = 1$, and let H be the entropy of this distribution. Then

$$(1/\log 3) \cdot H$$

is a lower bound on the weighted path length P_{opt} of an optimal binary search tree. This bound is sharp for infinitely many distributions.

Proof. Let T be any binary search tree. Define

$$L = \sum_{i=1}^n 3^{-(b_i+1)} + \sum_{j=0}^n 3^{-a_j}.$$

A simple induction argument shows that L is equal to 1, thus $\log L = 0$. Define

$$\beta'_i = 3^{-(b_i+1)} \quad \text{for } 1 \leq i \leq n$$

and

$$\alpha'_j = 3^{-a_j} \quad \text{for } 0 \leq j \leq n.$$

Then

$$\sum \beta'_i + \sum \alpha'_j = L = 1.$$

So $\alpha'_0, \beta'_1, \dots, \beta'_n, \alpha'_n$ is a frequency distribution. The following inequality is well known from coding theory (cf. Kameda and Weihrauch [3])

$$\sum \beta_i \log 1/\beta_i + \sum \alpha_j \log 1/\alpha_j \leq \sum \beta_i \log 1/\beta'_i + \sum \alpha_j \log 1/\alpha'_j.$$

It yields in our case

$$\begin{aligned} H &= \sum \beta_i \log 1/\beta_i + \sum \alpha_j \log 1/\alpha_j \\ &\leq \sum \beta_i \log 1/\beta'_i + \sum \alpha_j \log 1/\alpha'_j \\ &= \sum \beta_i \log 3^{(b_i+1)} + \sum \alpha_j \log 3^{a_j} \\ &= (\log 3) (\sum \beta_i (b_i + 1) + \sum \alpha_j a_j) \\ &= (\log 3) \cdot P \end{aligned}$$

with equality if and only if $\beta_i = \beta'_i$ and $\alpha_j = \alpha'_j$ for all i and j . Hence $(1/\log 3) \cdot H \leq P$.

Assume now that $\beta_i = \beta'_i$ and $\alpha_j = \alpha'_j$ for all i and j . Then the weights of the left and right subtree of any node in T are exactly equal. Hence Rule II will construct the tree T when applied to distribution $\alpha_0, \beta_1, \dots, \beta_n, \alpha_n$.

So

$$1/\log 3 \cdot H \leq P_{\text{opt}} \leq P_{\text{balanced}} = 1/\log 3 \cdot H$$

Hence the lower bound is sharp in this case.

q.e.d.

One of the referees pointed out to the author that Theorem 2 is in fact a special case of an information theoretic result due to Shannon. Specifically, every binary search tree corresponds to a ternary code tree derived by moving the weight of each node to a leaf extending from the node. Then the variable length coding theorem for ternary codes gives the same bound, $H/\log 3$.

As a corollary to Theorems 1 and 2 we obtain our main theorem.

Main Theorem. Let $\alpha_0, \beta_1, \dots, \beta_n, \alpha_n$ be any frequency distribution with $\sum \beta_i + \sum \alpha_j = 1$, let P_{opt} be the weighted path length of an optimal binary search tree, let P_{balanced} be the weighted path length of the tree constructed according to Rule II, and let H be the entropy of the distribution.

Then

$$\begin{aligned} 1/\log 3 \cdot H &\leq P_{\text{opt}} \leq P_{\text{balanced}} \leq 2 + (1 - \log(\sqrt{5} - 1))^{-1} \cdot H \\ 0.63 \cdot H &\leq P_{\text{opt}} \leq P_{\text{balanced}} \leq 2 + 1.44 \cdot H. \end{aligned}$$

Our main theorem clearly shows the importance of Rule II (or one of its modifications) as an approximation algorithm for constructing binary search trees. Furthermore, it exhibits a rather narrow interval for the weighted path length of optimal (or nearly optimal) binary search trees and thus gives a simple a priori test for the performance of binary search trees.

Acknowledgement. I am indebted to Prof. D. E. Knuth for reading a preliminary version of this paper, correcting an error in my Theorem 1 and suggesting some improvements.

Appendix

We only describe an implementation of Rule II. Rule II requires us to choose the root so as to equalize the total weight of the left and right subtrees as much as possible, i.e. one has to find i such that

$$\begin{aligned} & |(\alpha_0 + \beta_1 + \dots + \beta_{i-1} + \alpha_{i-1}) - (\alpha_i + \beta_{i+1} + \dots + \beta_n + \alpha_n)| \\ &= \min_{1 \leq j \leq n} |(\alpha_0 + \beta_1 + \dots + \beta_{j-1} + \alpha_{j-1}) - (\alpha_j + \beta_{j+1} + \dots + \beta_n + \alpha_n)| \end{aligned}$$

i can be found in time proportional to $\min(i, n-i+1)$ by searching for i simultaneously from both ends, i.e. by trying $i=1, i=n, i=2, \dots$ in that order. Having found i , one applies the algorithm recursively to $(\alpha_0, \beta_1, \dots, \beta_{i-1}, \alpha_{i-1})$ and $(\alpha_i, \beta_{i+1}, \dots, \beta_n, \alpha_n)$. This requires solving similar problems of size $i-1$ and $n-i$. Thus one obtains the following recurrence equation for $T(n)$, the time required to build a nearly optimal tree with n names by means of Rule II,

$$T(n) \leq \max_{1 \leq i \leq n} |T(i-1) + T(n-i) + c \min(i, n-i+1)|$$

and

$$T(0) \leq c$$

for some constant c . This inequality has a solution $T(n)$ with

$$T(n) \leq d(n \log n + 1).$$

$$\begin{aligned} T(n) &\leq \max_{1 \leq i \leq n} |T(i-1) + T(n-i) + c \min(i, n-i+1)| \\ &\leq \max_{0 \leq i \leq n/2} |T(i) + T(n-i-1) + c(i+1)| \\ &\leq \max_{0 \leq i \leq n/2} |d(i(\log i) + 1) + d((n-i-1) \log(n-i-1) + 1) + d(i+1)| \\ &\leq d \cdot \max_{0 \leq i \leq n/2} \left| 3 + (n-1) \log(n-1) \right. \\ &\quad \left. + (n-1) \left(\frac{i}{n-1} \log \frac{i}{n-1} + \frac{n-i-1}{n-1} \log \frac{n-i-1}{n-1} + \frac{i}{n-i} \right) \right| \\ &\leq d \cdot (n \log n + 1) + d(2 - \log(n-1)) \\ &\quad + d \cdot \max_{0 \leq i \leq n/2} (n-1) \left(\frac{i}{n-1} \log \frac{i}{n-1} + \frac{n-i-1}{n-1} \log \frac{n-i-1}{n-1} + \frac{i}{n-i} \right) \\ &\leq d(n \log n + 1) \end{aligned}$$

if $c \leq d$ and $n \geq 5$. Thus we only have to choose d large enough such that $T(n) \leq d(n \log n + 1)$ for $n \leq 4$.

References

1. Fredman, M. L.: Two applications of a probabilistic search technique: Sorting $X + Y$ and building balanced search trees. 7th ACM Symposium on Theory of Computing, Albuquerque, 1975
2. Hu, T. C., Tan, K. C.: Least upper bound on the cost of optimum binary search trees. *Acta Informatica* 1, 307–310 (1972)
3. Kameda, T., Weihrauch, K.: Einführung in die Kodierungstheorie I. BI Skripten zur Informatik, Vol. 7. Mannheim: Bibliographisches Institut 1971
4. Knuth, D. E.: Optimum binary search trees. *Acta Informatica* 1, 14–25 (1971)
5. Knuth, D. E.: The art of computer programming, Vol. 3. Reading (Mass.): Addison-Wesley 1973
6. Rissanen, J.: Bounds for weighted balanced trees. *IBM J. Res. Develop.* March 1973, 101–105
7. Walker, W. A., Gotlieb, C. C.: A top-down algorithm for constructing nearly optimal lexicographical trees, in *Graph theory and Computing*. New York: Academic Press 1972

Kurt Mehlhorn
Fachbereich 10
Universität des Saarlandes
D-6600 Saarbrücken
Federal Republic of Germany