Top-down Algorithms for Constructing Nearly Optimal Binary Search Trees

Yasuyuki NISHIKAWA,* Yuuji YOSHIDA** and Teruo FUKUMURA***

1. Introduction

A binary search tree is an important technique for organizing large files. There are several conventional methods for constructing an optimal (or a nearly optimal) binary search tree. But these methods are time-consuming or space consuming and so not applicable to practical problems involving several hundreds of keys.

In this paper, we present some methods for constructing a nearly-optimal binary search tree. Our algorithms consist of heuristic top-down methods and Knuth's "Algorithm K" (dynamic programming method)[1] and take time proportional to $N^{3/2}$ (N is the number of keys) and storage proportional to N.

Experimental results show good performances of our methods. The nearly-optimal trees can be expected to suppress the increase of average search length within 0.1 % of that of optimal tree.

2. Preliminaries

In this section, we give several definitions.

[Definition 1] A binary tree (b.t.) on a set $V=\{v_1, v_2, \ldots, v_n\}$ is defined as follows:

(1) ϕ (null set) is a b.t.

(2) Let T_L and T_R be two b.t.s, then for $\forall v \in V$, (T_L, v, T_R) is a b.t.

(3) Nothing else is a b.t.

[Definition 2] For a b.t. $T=(T_L, v, T_R)$,

root(T)=v, left(T)=T_L, right(T)=T_R, $\mathcal{N}(T)$= a set of nodes of T,

|T| = the number of nodes of T, $1_T(k)$=level of node K in the tree T.

[Definition 3] Let $K=\{K_1, K_2, \ldots, K_N\}$ be a totally ordered set and $\prec$ is its ordering relation. Then a b.s.t. for K is a b.t. T such that $\mathcal{N}(T)=K$, and for all subtree T' of T, $\forall K_i \in \mathcal{N}(\text{left}(T'))$, $\forall K_j \in \mathcal{N}(\text{right}(T'))$ are in the relation,

---

This paper first appeared in Japanese in Joho-Shori (Journal of the Information Processing Society of Japan), Vol. 17, No. 8 (1976), pp. 736~742.

* TOYOTA MOTOR Co., Ltd.
** Computation Center, Nagoya University
*** Faculty of Engineering, Nagoya University

$K_i \prec \text{root}(T') \prec K$

[Definition 4] $\mathbb{K}(i,j)$ denotes $\{K_m \mid i \leq m \leq j\}$, and $\mathbb{T}(i,j)$ denotes the set of all b.s.t. for $\mathbb{K}(i,j)$. $T(i,j)$ denotes an element of $\mathbb{T}(i,j)$. $\mathbb{K}(1,N)$ is denoted only $\mathbb{K}$.

[Definition 5] Let $\{p_1, p_2, \ldots, p_N\}$ be the probability distribution of $\mathbb{K}$. Then, the cost of a b.t. T for $\mathbb{K}(i,j)$ is defined as $c(T) = \sum_{\theta=i}^{j} p_\theta (l_T(\theta)+1)$

[Definition 6] The cost of the optimal b.s.t. for $\mathbb{K}(i,j)$ is defined as

$$c(i,j) = \min_{T \in \mathbb{T}(i,j)} c(T)$$

When a set of keys $\{K_i, K_{i+1}, \ldots, K_j\}$ is given, some key K_m will be selected as the root of the optimal b.s.t. for $\mathbb{K}(i,j)$ depending on the following conditions;

(1) P_m is large.

(2) The difference between the weights of T(i,m-1) and T(m+1,j) is small.

(3) The difference between the depths of T(i,m-1) and T(M+1,j) is small.

We define two measures to represent the reasonability of these conditions about $T=(T_L(k), K_k, T_R(k))$.

[Definition 7] (weight balance σ of T)

$$\sigma_k = \min\left(\frac{\omega_L(k)+p_k}{\omega+p_k}, \frac{\omega_R(k)+p_k}{\omega+p_k}\right)$$

where, $\omega_L(k) = \sum_{\theta=i}^{k-1} p_\theta$, $\omega_R(k) = \sum_{\theta=k+1}^{j} p_\theta$, $\omega = \sum_{\theta=i}^{j} p_\theta$

[Definition 8] (depth balbnce τ of T)

$$\tau_k = \min\left(\frac{1+d_L(k)}{d}, \frac{1+d_R(k)}{d}\right)$$

where $d_L(k) = \log_2(k-i+1)$, $d_R(k) = \log_2(j-k+1)$ and $d = \log_2(j-i+2)$

Here, we define two heuristic functions $\mathcal{H}_1, \mathcal{H}_2$ which measures how much K_k is likely to be the root of an optimal b.s.t. for $\mathbb{K}(i,j)$.

[Definition 9] For $T=(T_L(k), k, T_R(k))$, $\mathcal{H}_1(k) = p_k \times \sigma_k$, $\mathcal{H}_2(k) = p_k \times \sigma_k \times \tau_k$

3. Algorithms for Constructing Nearly-Optimal Binary Search Trees

In this section, we first define some notation and then present a fundamental theorem which gives a theoretical base to the algorithms described later

[Definition 10] Let p_i be the probability that $K_i \in \mathbb{K}$ is retrieved. Let $\mathcal{E}_0$ be $\{K \mid K \prec K_1\}$, $\mathcal{E}_N$ be $\{K \mid K_N \prec K\}$, and generally $\mathcal{E}_i$ be $\{K \mid K_i \prec K \prec K_{i+1}\}$ $(i=1,2,\ldots,N-1)$. Let q_i be the probability that a key belonging to $\mathcal{E}_i$ is retrieved. Then, an N-L b.s.t. for $\mathbb{K}$, $\{p_i\}, \{q_i\}$ is a b.t. $\mathcal{T}$ which satisfies the following conditions:

(1) $\mathcal{T}$ has 2·N+1 nodes and each node has either no subtree of just two subtrees.

(2) $\mathcal{T}$ is a b.s.t. for a set of keys $\mathbb{K} = \{\mathcal{E}_0, \{K_1\}, \mathcal{E}_1, \{K_2\}, \ldots, \{K_N\}, \mathcal{E}_N\}$.

The ordering relation on $\mathbb{K}$ is defined as the order in which elements are listed

above.

[Definition 11] The cost of b.s.t. $\mathcal{J}$, $c(\mathcal{J})$, is defined as follows:

$$c(\mathcal{J})=\sum_{\eta=i+1}^{j} p_\eta(1_{\mathcal{J}}(\eta)+1)+\sum_{\xi=i}^{j} q_\xi 1_{\mathcal{J}}(\xi)$$

[Definition 12] Let $\{\mathcal{J}_1,\mathcal{J}_2,\ldots,\mathcal{J}_M\}$ $(\mathcal{J}_m \prec \mathcal{J}_{m+1})$ be a subset of the set $\{K_i,K_{i+1},\ldots,K_j\}$. Then we can construct an N-L b.s.t. for these M keys and define two sets of nodes of this tree, INT and EXT, where $INT=\{\mathcal{J}_m \mid 1\leq m\leq M\}$, $EXT=\{\mathcal{E}_m \mid 0\leq m\leq M\}$.

If a set of keys $\{K_i,K_{i+1},\ldots,K_j\}$ is given, then we can construct a b.s.t. T and an N-L b.s.t. $\mathcal{J}$ for some INT and EXT. The relation between c(T) and $c(\mathcal{J})$ is given as: $c(T)=c(\mathcal{J})+\sum_{m=0}^{M} c(T\{\mathcal{E}_m\})$, where $T\{\mathcal{E}_m\}$ denotes a b.s.t. for $\mathcal{E}_m$.

Now, we have a fundamental theorem for constructing algorithms.

[Theorem] If, in the configuration of T^*, there is no element of EXT on the paths from the root to the elements belonging to INT, then the configuration of $\mathcal{J}^*$ coincides with that of T^* for the part pf trees corresponding to INT.

The algorithm is presented as a recursive procedure to construct an optimal b.s.t. $T^*(i,j)$ for K(i,j) and is described as A(i,j). In the algorithm, "Algorithm K" denotes Knuth's algorithm by D.P.

[Algorithm A(i,j)]

step 1. $n=j-i+1$, if $n\leq f(n)$, then go to step 2, else go to step 3.

step 2. (1) if n=0, then $T^*(i,j)=\phi$ and algorithm terminates.

(2) if n=1, then $T^*(i,j)=(\phi,k_i,\phi)$ and algorithm terminates.

(3) if $n\geq 2$, then construct $T^*(i,j)$ by Algorithm K, and algorithm terminates.

step 3. Select f(n) keys from K(i,j) by algorithm SELECT or algorithm CHOICE and let those nodes corresponding to the keys be INT and other nodes be EXT.

step 4. Construct an optimal N-L b.s.t. for INT and EXT by Algorithm K. Let K_{k^*} be the root of the optimal tree.

step 5. Construct two optimal b.s.t. $T^*(i,k^*-1)$ and $T^*(k^*+1,j)$ by algorithm $A(i,K^*-1)$ and $A(k^*+1,j)$.

step 6. Let $(T^*(i,k^*-1),K_{k^*},T^*(k^*+1,j))$ be the optimal b.s.t. for K(i,j) by this algorithm.

From this general algorithm, we derive three algorithms, Algorithm 1, Algorithm 2 and Algorithm 3 by setting f(n)=M(constants), f(n)=F(n) (F is a simple monotonic increasing function and $1\leq F(n)\leq n$), and $f(n)=\max(F(n),M_\perp)$ respectively.

Two subalgorithms, SELECT and CHOICE are as follow.

[Algorithm SELECT]

step 1. Compute $\mathcal{K}(k)$ for k=i,i+1,...,j.

step 2. Let INT be the m keys out of $\{K_i, K_{i+1}, \ldots, K_j\}$ that have larger values of $\mathcal{K}(k)$.

[Algorithm CHOICE]

step 1. INT $=\phi$, $\mathcal{K}=\{K(i,j)\}$ and P=1. Let $K_1, K_2, \ldots$ denote elements of $\mathcal{K}$.

step 2. p=1, $\tilde{\mathcal{K}}=\phi$

step 3. Select the key out of K_p which has the largest value of $\mathcal{K}$ and let K_t denote the key. INT $\leftarrow$ INT $\cup \{K_t\}$. Partition K_p into two subsets, one is a set whose element is smaller than K_t and another is a set whose element is larger than K_t. Let these set be elements of $\tilde{\mathcal{K}}$ if they are not null set.

step 4. If the number of elements of INT is m, then algorithm terminates. Otherwise,

(i) if p<p then p$\leftarrow$p+1 and go to step 3.

(ii) if p=p then $\mathcal{K} \leftarrow \tilde{\mathcal{K}}$, P$\leftarrow$the number of elements of $\tilde{\mathcal{K}}$ and go to step 2.

4. Evaluation of Required Amounts of Storage and Computational Time.

Let M be the number of Keys. Algorithm K requires $O(M^2)$ storage for M keys. Then, all of three algorithms requires $O(N)$ storage when $M \leq \sqrt{2N}$, where M is the parameter of algorithm 2. On the othehand, Table 1 shows time requirements of the algorithms in several cases.

Table 1. Time Complexity of Each Algorithms

Alg. \ INT	SELECT	CHOICE
1	$O(N^{3/2}\log_2 N)$	$O(N^{3/2})$
2	$O(N^{3/2})$	$O(N(\log_2 N)^2)$
3	$O(N^{3/2})$	$O(N^{4/3})$ $(N \gtrsim 10^6)$
		$O(N\log_2 N)^2)$ $(N \lesssim 10^6)$

(N: Number of keys)

5. Experimental Results and Remarks

Our algorithms were applied to many sets of keys with frequency distribution generated according to Zipf's law with respect to the frequency distribution of the set of English words. Table 2. shows the degradation of the cost in nearly-optimal b.s.t. v.s. optimal b.s.t. The degradation ΔC is defined by

$$\Delta C = \frac{c(T)-c(T^*)}{c(T^*)} \times 100\ \%$$

where c(T) is the cost of the nearly-optimal tree T, and c(T*) is that of the optimal one. Fig. 1 shows the performances of algorithm 1,2 and 3 by the frequency distribution of ΔC, revealing that the performances of our algorithms are more akin to the optimality than conventional methods[2),3)]. Fig. 2 shows the

performance with respect to the computational time. It validates the estimate of the time complexity described in 4. From these two figures we can see that Algorithm 2 is the best of the three with respect to the computation time, which algorithm 3 is the best for the nearly-optimality.

Acknowledgement

Authors would like to thank Prof. Namio Honda for his invaluable advices. They would also like to thank the members of Fukumura laboratory for their valuable discussion.

References

1) D.E.Knuth: The Art of Computer Programming, Vol. 3, Addison-Wesley, (1973)

2) J. Broun and E. G. Coffman: Nearly Optimal Binary Search Trees, IFIP Congress 71,

3) W.A. Walker and C. C. Gotlieb: A Top-down Algorithm for Constructing Nearly Optimal Lexicographic Trees, Graph Theory and Computing (ed. by R. C. Read), Academic Press (1972)

Table 2. Comparisons between Nearly-Optimal b.s.t. and the optimal b.s.t.

INT	SELECT		CHOICE	
H				
N=100	0.06 + 0.18 0.38 + 0.41 0.28 + 0.38	0.07 + 0.20 0.49 + 0.46 0.36 + 0.40	0.11 + 0.24 0.35 + 0.35 0.27 + 0.32	0.11 + 0.33 0.23 + 0.38 0.21 + 0.38
N=200	0.05 + 0.14 0.34 + 0.31 0.21 + 0.28	0.07 + 0.17 0.43 + 0.33 0.27 + 0.29	0.08 + 0.17 0.29 + 0.24 0.18 + 0.23	0.03 + 0.08 0.14 + 0.12 0.10 + 0.11

(Each entry shows 'average + deviation'.)

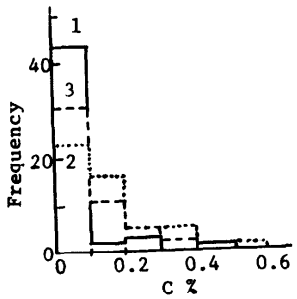

Fig.1 Performance of Algorithm 1,2 and 3 (CHOICE/$\mathcal{N}_2$:N=200)

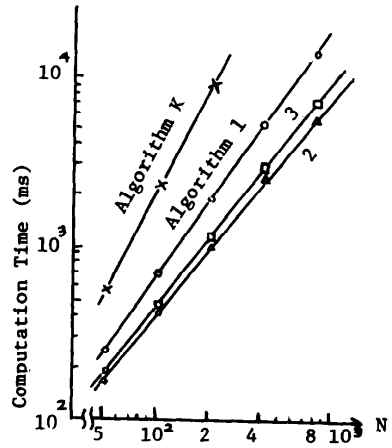

Fig.2 Computation Time of Algorithm 1,2 and 3 (CHOICE/$\mathcal{N}_2$)