INFORMATION AND CONTROL 63, 217–225 (1984)

Hash Functions for Priority Queues

M. AJTAI, M. FREDMAN, AND J. KOMLÓS*

University of California San Diego,
La Jolla, California 92093

The complexity of priority queue operations is analyzed with respect to the cell probe computational model of A. Yao (*J. Assoc. Comput. Mach.* **28**, No. 3 (1981), 615–628). A method utilizing families of hash functions is developed which permits priority queue operations to be implemented in constant worst-case time provided that a size constraint is satisfied. The minimum necessary size of a family of hash functions for computing the rank function is estimated and contrasted with the minimum size required for perfect hashing.

INTRODUCTION

The following data structure problem is considered. Given a fixed universe of integers $U = \{1, 2, ..., m\}$ we wish to represent a subset S in a manner permitting efficient processing of the following operations (x is an arbitrary element of U):

Insert(x)
$S \leftarrow S \cup \{x\}$
Delete(x)
$S \leftarrow S - \{x\}$
Query(x)
Return $(\text{rank}(x), x_{\text{pred}}, x_{\text{succ}})$
where
$\text{rank}(x) =$ the number of elements in S which are $\leqslant x$
$x_{\text{pred}} =$ the largest element of S which is $< x$ (Λ if it does not exist)
$x_{\text{succ}} =$ the smallest element of S which is $\geqslant x$ (Λ if it does not exist)

Observe that Query (1) returns the smallest of S. Therefore, with these operations we can readily implement priority queues, range queries, and closest neighbor queries.

* Supported in part by the National Science Foundation Grant MCS-8204031.

0019-9958/84 $3.00

Our primary interest concerns the complexity of these operations with respect to the "cell probe" model of computation introduced by Yao (1981). In that paper, Yao conjectures that priority queue operations cannot be executed in constant time in the cell probe model "even if arbitrary encoding is allowed." Our first result, however, shows that the operations in (1) can each be executed with at most four probes (provided that m is large enough in comparison with $|S|$). The method we use is similar to Yao's method for answering membership queries in two probes. Briefly, a query is executed by probing an initial memory cell, say cell(0), and then, based on the contents c of cell(0) and the input x, a second cell is probed which contains an item in the subset S. This item y is then examined in conjunction with c and x to determine the rank of x. We find the predecessor or the successor with one more probe. When executing an insertion (deletion), we proceed as if doing a query for the rank, then insert (remove) the input into (from) an appropriate cell, and then modify the contents of cell(0). The amount of space used is $|S|+1$. The details are provided below.

It is useful to view the above data structure from another perspective. Associated with the family of subsets of U of size n is a family of hash functions $F=\{h_1,..., h_r\}$, $|F| \leqslant |U| = m$. The domain of the functions h_j is U and the range is $\{1, 2,..., n\}$. In representing a particular subset S, $|S| = n$, a hash function $h_k \in F$ is chosen and the index k of h_k is stored in cell(0). The following conditions concerning F and the choice h_k associated with S hold:

1. h_k is a perfect hash function for S and

2. in determining rank(x), it suffices to have knowledge of x, k, and the unique element y in S such that $h_k(x) = h_k(y)$.

Each y in S is stored in cell($h_k(y)$).

Melhorn [3] posed the problem of determining the minimum size $\Theta_{m,n}$ of a family of functions satisfying condition 1 only. This relates to the minimum possible program size of perfect hash functions. The best known estimate for $\Theta_{m,n}$ is provided in Fredman and Komlós [1]. Namely, $\Theta_{m,n} \approx e^n \log m$. This estimate can be regarded as a measure of the complexity of the membership query problem since perfect hashing is naturally identified with membership queries.

We now pose the analogous question; what is the minimum size $\phi_{n,m}$ of a family F of functions satisfying both conditions 1 and 2? We regard this as a measure of complexity for rank queries. Our second result states that for fixed n, $c_1(\log m)^{n-1} < \phi_{n,m} < c_2(\log m)^{n-1}$ for large m, where c_1 and c_2 are positive constants depending only on n. Therefore, we are tempted to classify membership queries as having difficulty $\log m$ (fixed n, large m) and classify rank queries as having difficulty $(\log m)^{n-1}$ (fixed n, large m).

Recently, Ajtai has shown that given polynomial space n^c, and arbitrary k, there is a value for m for which no data structure can solve rank queries with only k probes. So our first result, which shows that these queries can be answered in constant time (with linear space) when m is sufficiently large cannot be extended to intermediate values of m. This provides even more striking evidence for the difference in difficulty between membership and rank queries, since Fredman, Komlós, and Szemerédi [2] shows that membership queries can be solved in constant time (with linear space) independently of the relationship between m and n.

The Data Structure

We represent the elements of U in binary. A hash function for the set S will be represented using a trie. A trie is a binary tree with a digit position specified in each internal node. The digit of a node is a more significant digit than those of its descendants. With an internal node (with digit j, say) there corresponds a subset V of S. (For the root, $V = S$.) Corresponding to the left son is the non-empty subset V_0 of V consisting of elements with zero in the jth digit; with the right son the non-empty subset V_1 of elements with 1 in the jth digit. Of special importance is the condition that we choose j to be the most significant digit which distinguishes between two elements of V. In each leaf is stored a number from 1 to n: the address of the memory cell containing the single element associated with that leaf.

In executing Query(x), we find the leaf of the trie determined by x (in matching the bit values in the digit positions specified by the path). Next, we access the contents y of the memory cell whose address is stored in the leaf. Observe that y is an element of S which matches x in the digit positions specified by the path. A little reflection shows that rank(x) is uniquely determined by x, y, and the trie (see the example and figures below). Furthermore, the addresses of the successor and predecessor are clearly determined by rank(x) and the trie. Note that the number of possible tries, as we have described them, is at most $Z_{m,n} = T_n \cdot n! \cdot (\log m)^{n-1}$, where T_n is the number of binary trees with n leaves. Moreover, for $m \geqslant e^{cn \log n}$, $Z_{m,n} \leqslant m$. Therefore, when m is sufficiently large compared to n, the trie associated with a set S can be specified by an integer from 1 to m, which we store in cell(0). (Yao's cell probe model restricts the contents of a memory cell to be an integer from 1 to m.)

In executing an insertion, the new item is placed in an available cell (which can be determined from the trie), and the trie is modified to appropriately reflect the change. Deletions are handled similarly.

Example. Figure 1a shows the set $S = \{a, b, c, d\}$ and its representation

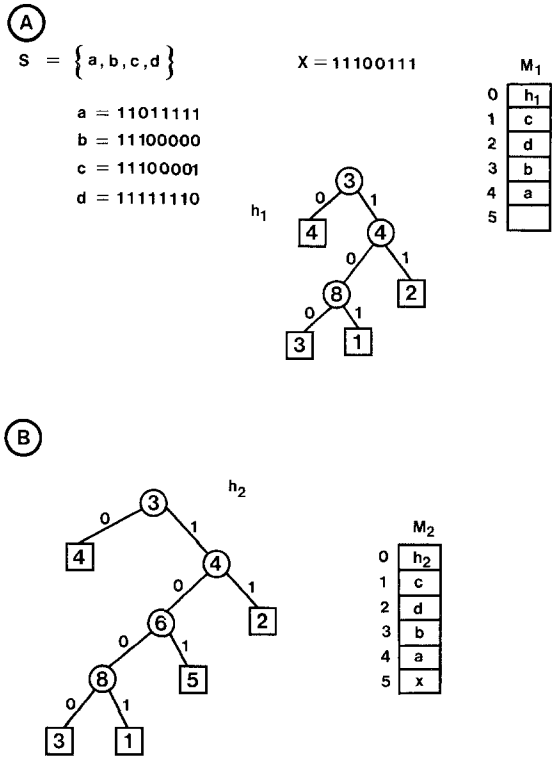

FIGURE 1

M_1 in memory using hash function h_1. In executing Query(x) (for x as shown in Fig. 1a) we find that $h_1(x) = 1$. Comparing x with the contents c of cell($h_1(x)$), we find that the first digit in which x and c differ is the 6th, wherein x has a 1 and c has a 0. With this information we can deduce, by examining the trie h_1, that rank(x) = 3. Figure 1b shows the result of inserting x (placing it in cell(5)), yielding a new hash function h_2 and representation M_2.

LOWER BOUND

The lower bound we prove is slightly more general than the result described in the Introduction. A *hash function* $G = (h, f)$ consists of a function h mapping the universe $U = \{1,..., m\}$ into $\{1,..., n\}$ and a 2-variable function $f(x, y)$, $x, y \in U$, whose range is the space of answers appropriate for the query type (e.g., $\{0,..., n\}$ for rank queries). A hash function G *accommodates* a subset S with respect to a query function $Q(x)$

provided that there is a way to store the n elements of S into n memory cells; cell(1),..., cell(n), so that for each x in U, $Q(x)=f(x, y)$, where y is the contents of cell($h(x)$).

Assume now that $Q(x)=\text{rank}(x)$. A specific G can accommodate many subsets of size n (even a positive fraction of the $\binom{m}{n}$ subsets), but we will show that it can represent only a small proportion with respect to the (highly distorting) measure defined below. Hence, many hash functions are required to accommodate *all* subsets S of size n. Our lower bound result is provided by the following theorem.

THEOREM. *If a family $J=\{G_1,..., G_r\}$ of hash functions accommodates all of the $\binom{m}{n}$ subsets, then $|J|=\Omega((\log m)^{n-1})$.*

DEFINITION. The weight of an ordered set $S=\{a_1,..., a_n\}$ is defined to be $w(S)=\prod_{i=1}^{n-1}(a_{i+1}-a_i)^{-1}$.

Remark. The following interesting interpretation can be given for our measure. It is concentrated on subsets having the following characteristic. When peering at the set through a magnifying glass of any strength, you can see at most two points; points are either too close to be distinguished, or beyond the field of view.

Proof of the Theorem. Observe that the total weight of the family of all $\binom{m}{n}$ subsets is $\Theta(m(\log m)^{n-1})$ (the constants involved depend on n only). We show that $W(G)$, the total measure of the subsets accommodated by a hash function G, satisfies

$$W(G)=O(m). \tag{2}$$

Our theorem is an easy consequence of this fact.

We now proceed to establish (2). Fix $G=(h, f)$ and let R denote the family of sets accommodated by G. We say that a set S in R is *faithfully* represented provided that each element a_i in S is stored in cell($h(a_i)$), otherwise S is *unfaithfully* represented. Below we show that the total weight of the faithfully represented sets is $O(m)$. Assuming this for the moment, we now show that the total weight of all sets in R is also $O(m)$, implying out theorem.

We argue by induction on n. Let R_f denote the family of faithfully represented sets in R, and for each 4-tuple (i, j, k, p) $1\leqslant i, j, k, p\leqslant n$, $i\neq j$, let R_{ijkp} denote the following family of sets in R. A set $S=\{a_1,..., a_n\}$, $a_1<\cdots<a_n$, in R belongs to R_{ijkp} provided that $h(a_i)=k$, a_j is stored in

cell(k) and a_i is stored in cell(p). Clearly each unfaithfully represented set in R belongs to some R_{ijkp} so that

$$R = R_f \cup \bigcup_{\substack{i,j,k,p \\ i \neq j}} R_{ijkp}.$$

We show that the total weight of each R_{ijkp} is $O(m)$ from which it follows that the total weight of R is $O(m)$ since the number ($<n^4$) of R_{ijkp} families is independent of m.

Now fix i, j, k, p. We demonstrate that for each set S in R_{ijkp} the value of a_j uniquely determines the value for a_i. For each x in $U = \{1,..., m\}$ such that $h(x) = k$, $\text{rank}(x) = f(x, a_j)$, since a_j is stored in cell(k). Clearly, $a_i = \min\{x \mid h(x) = k \text{ and } f(x, a_j) = i\}$ since a_i is the smallest element of U of rank i and $h(a_i) = k$. Thus a_i is uniquely determined by a_j as claimed. Now let Q be the family of sets $S' = \{a_1,..., \hat{a}_i,..., a_n\}$ ($\hat{a}_i$ denotes that a_i has been deleted) such that $S = \{a_1,..., a_i,..., a_n\}$ belongs to R_{ijkp}. We claim that the sets in Q can be represented by a single hash function G' for subsets of size $n-1$. To begin with, we store $\{a_1,..., \hat{a}_i,..., a_n\}$ in the same manner as $\{a_1,..., a_n\}$ except that cell(p) (in which a_i would have been stored) is removed from memory.

Before defining G', we first define $\hat{G} = (\hat{h}, \hat{f})$ as follows:

$$\begin{aligned} \hat{h}(x) &= k && \text{if } h(x) = p, \\ &= h(x), && \text{otherwise;} \\ \hat{f}(x, y) &= f(x, a_i), && \text{if } h(x) = p, \\ &= f(x, y), && \text{otherwise.} \end{aligned}$$

(Observe that the a_i in $f(x, a_i)$ is well defined; when $h(x) = p$ we know that $y = a_j =$ contents of cell($\hat{h}(x)$), and from a_j the value of a_i is uniquely determined as described above. In other words, a_i is a function of y.) Using $\hat{G}$, we can determine rank(x) relative to the set $\{a_1,..., a_i,..., a_n\}$. We then easily deduce rank(x) relative to $\{a_1,..., \hat{a}_i,..., a_n\}$. More formally, we define $G' = (h', f')$ as follows:

$$\begin{aligned} h'(x) = \hat{h}(x) \quad \text{and} \quad f'(x, y) &= \hat{f}(x, y), && \text{if } \hat{f}(x, y) < i, \\ &= \hat{f}(x, y) - 1, && \text{if } \hat{f}(x, y) \geq i. \end{aligned}$$

This G' represents the sets in Q. Because $w(S) = O(w(S'))$ (for S in R_{ijkp} and $S' = S - \{a_i\}$ in Q) and because the total weight of Q is $O(m)$ by the induction hypothesis, we conclude that the total weight of R_{ijkp} is $O(m)$.

To complete the proof, we must show that the total weight of R_f is $O(m)$. Given a set $S = \{a_1,..., a_n\}$, $a_1 < \cdots < a_n$, in R_f, we refer to the subset

$\{a_i, a_{i+1},..., a_j\}$ as an (i, j)-*interval* of S, and we define its weight to be $\prod_{m=i}^{j-1}(a_{m+1}-a_m)^{-1}$. (The weight of a singleton set is defined to be 1.) We argue by induction on the size $\sigma = j-i+1$ of (i, j)-intervals that the total weight of the (i, j)-intervals of the sets in R_f is $O(m)$. The case $\sigma = n$ implies our theorem.

We actually prove a slightly stronger statement. We say that an (i, j)-interval $\{a_i,..., a_j\}$ is an (i, j, c, d)-*interval* provided that $c \leqslant a_i$ and $a_j \leqslant d$. We show that the total weights of the (i, j, c, d)-intervals of the sets in R_f is $O(d-c)$. The case $\sigma = 1$ is obvious. Assume that $\sigma > 1$, fix (i, j) with $\sigma = j-i+1$, and let $Q = Q(i, j, c, d)$ denote the set of (i, j, c, d)-intervals of the sets in R_f. Next, for $i \leqslant r < j$ and $1 \leqslant k \leqslant n$, let $Q_{r,k}$ denote the family containing the sets $S = \{a_i,..., a_j\}$ in Q such that

(a) The gap from a_r to a_{r+1} is the largest gap in S (leftmost if there is more than one such gap), and

(b) $h(a_r) = k$.

The families $Q_{r,k}$ partition Q. We shall show that the total weight of each $Q_{r,k}$ is $O(d-c)$. Let H_k denote the subset $\{x \mid h(x) = k\}$. Since $\text{rank}(x) = f(x, a_r)$ for each x in H_k (because of faithfulness), each value $v = a_r$ determines a unique interval $(y, z]$ such that for every set S in $Q_{r,k}$ with $a_r = v$, the value of a_{r+1} is confined to the interval $(y, z]$. Namely, $y = \max\{x \mid x \in H_k \text{ and } f(x, v) = r\}$ and $z = \min\{x \mid x \in H_k \text{ and } x > y\}$. (We pick $z = d$ if there is no x in H_k larger than y.) Observe that $y \geqslant a_r$ since $a_r = v$ lies in the set in terms of which y is defined. Now let L denote the family of sets in $Q_{r,k}$ such that $z-y < y-a_r$ and let B denote the family of remaining sets (satisfying $z-y \geqslant y-a_r$). We consider L and B separately and show that they have total weights $O(d-c)$.

The total weight of L is given by

$$\sum_v \sum_{\substack{S \in L \\ a_r = v}} w(S) = \sum_v \sum_{\substack{S \in L \\ a_r = v}} w(S_1)\, w(S_2)/(a_{r+1}-a_r) \tag{3}$$

where $S_1 = \{a_i,..., a_r\}$ and $S_2 = \{a_{r+1},..., a_j\}$. Since $a_{r+1} \in (y, z]$, the condition defining L implies that

$$(z-a_r)/2 \leqslant a_{r+1}-a_r \leqslant z-a_r, \tag{4}$$

and therefore, the sum in (3) is bounded by

$$2 \sum_v \sum_{\substack{S \in L \\ a_r = v}} w(S_1)\, w(S_2)/(z-a_r). \tag{5}$$

Since the gap from a_r to a_{r+1} is a maximal gap in S, the elements of S_2 are confined to the interval $(a_r, a_r + n(z - a_r))$. Thus, for fixed v, the sets S_2 appearing in the inner sum in (5) are $(r+1, j, v, v + n(z-v))$-intervals, and (using the induction hypothesis) $\sum_{S \in L, a_r = v} w(S_2) = O(z - v) = O(z - a_r)$. Thus, we can bound (5) by

$$O\left(\sum_{v} \sum_{\substack{T \\ T = S_1 \text{ for} \\ \text{some } S \text{ in } L \\ a_r = v}} w(T)\right) = O\left(\sum_{\substack{T \\ T = S_1 \text{ for} \\ \text{some } S \text{ in } L}} w(T)\right). \tag{6}$$

The latter sum in (6) involves sets which are (i, r, c, d)-intervals, and therefore (using the induction hypothesis) is bounded by $O(d - c)$. This completes the analysis of L.

The intervals $(y, z]$ associated with the values of a_r (from here-on we write $a_r \to (y, z]$ to indicate this association) are non-overlapping since their endpoints are consecutive values of H_k. In analyzing the total weight of B, we express this quantity as

$$\sum_{(y,z]} \sum_{\substack{S \in B \\ a_r \to (y,z]}} w(S). \tag{7}$$

The lemma below shows that the inner sum in (7) is bounded by $O(z - y)$. Evaluating the outer sum then gives $O(d - c)$ since the $(y, z]$ involved in the sum are disjoint sub-intervals of $[c, d]$. This completes the analysis of B and the proof of the theorem.

LEMMA. *For a fixed* $(y, z]$, $\sum_{S \in B, a_r \to (y,r]} w(S) = O(z - y)$.

Proof. Since (by the definition of B), $z - y \geqslant y - a_r$, we have that $a_{r+1} - a_r \leqslant 2(z - y)$. Moreover (using the notation from the analysis of L), for each S such that $a_r \to (y, z]$ we observe that y separates S_1 and S_2 $(a_r \leqslant y < a_{r+1})$. Now consider those S such that $2^p \leqslant a_{r+1} - a_r < 2^{p+1}$. Since (a_r, a_{r+1}) defines a maximum gap, S_1 is an $(i, r, y - n \cdot 2^{p+1}, y)$-interval and S_2 is an $(r+1, j, y, y + n \cdot 2^{p+1})$-interval. Summing over these S, we obtain

$$\begin{aligned} \sum_S w(S) &\leqslant \sum_S w(S_1)\, w(S_2)/(a_{r+1} - a_r) \\ &\leqslant \sum_{\substack{T \\ T = S_1 \text{ for} \\ \text{some } S}} w(T) \cdot \sum_{\substack{T \\ T = S_2 \\ \text{for some } S}} w(T)/2^p = O(2^p) \end{aligned}$$

(by applying our induction hypothesis to each sum in the product).

Finally, we sum over p such that $1 \leqslant 2^p \leqslant 2(z-y)$, obtaining $O(z-y)$ as promised.

Open Question. We conclude by stating the following problem. Our query algorithm is not very realistic, as searching through a trie requires more than constant time in actuality. Note, however, that only $O(n \log\log m)$ bits of the possible $\log m$ bits of cell(0) are utilized. By utilizing all $\log m$ bits, one might hope for a family of hash functions which are more readily evaluated.

RECEIVED May 29, 1984; ACCEPTED March 20, 1985

REFERENCES

FREDMAN, M., AND KOMLÓS, J. (1984), On the size of separating systems and families of perfect hash functions, *SIAM J. Algebraic and Discrete Methods* **5**. No. 1 61–68.

FREDMAN, M., KOMLÓS, J., AND SZEMERÉDI, E. (1982), Storing a sparse table with $O(1)$ worst case access time, *in* Proceeding of the 23rd Ann. Sympos. on Found. of Comput. Sci."

MELHORN, K. (1982), On the program size of perfect and universal hash functions, *in* "Proceedings of the 23rd Ann. Sympos. on Found. of Comput. Sci."

YAO, A. Should tables be sorted, *J. Assoc. Comput. Mach.* **28**, No. 3 (1981), 615–628.