Storing a Sparse Table with $O(1)$ Worst Case Access Time

MICHAEL L. FREDMAN AND JÁNOS KOMLÓS

University of California, San Diego, La Jolla, California

AND

ENDRE SZEMERÉDI

University of South Carolina

Abstract. A data structure for representing a set of n items from a universe of m items, which uses space $n + o(n)$ and accommodates membership queries in constant time is described. Both the data structure and the query algorithm are easy to implement.

Categories and Subject Descriptors: E.1 **[Data Structures]**: Tables; F.2.2 **[Analysis of Algorithms and Problem Complexity]**: Nonnumerical Algorithms and Problems—*sorting and searching*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Hashing, complexity, sparse tables.

1. *Introduction*

The following searching problem is considered. A set S of n distinct names from a universe U of m possible names ($m \geq n$) is to be stored in a memory T in a manner permitting efficient processing of membership queries of the form, "Is q in S, and if so, then where can it be found in T?" We assume that the set S is static, so that our main concerns are the storage required for S and the time required for processing queries. Hashing schemes provide a solution to this problem utilizing $O(n)$ storage and permitting queries in $O(1)$ average time per query. In this paper, we concentrate on the worst case time required for a query, while retaining an $O(n)$ bound on storage. This question has been considered in various papers, for example, [1]–[4]. Yao [4] proposes an interesting complexity model for this problem. In Yao's framework, S is a subset of $U = \{1, 2, \ldots, m\}$ of cardinality n. The memory T stores an item from U in each of its cells, and these cells can be randomly accessed by address. A query for an element q in U is processed by probing a sequence of cells in T. This sequence of probes can be adaptive: The

This material is based upon work supported in part by National Science Foundation Grants MCS 76-08543, MCS 82-04031, and MCS 79-06228.

Authors' Addresses: M. L. Fredman and J. Komlós, Department of Electrical Engineering and Computer Science, University of California, San Diego, Mail Code C-014, La Jolla, CA 92093; E. Szemerédi, Mathematical Institute, PF428, 1395 Budapest, Hungary.

next cell to be probed is determined precisely by q and the contents of the cells previously probed. Query time is measured in terms of the number of cells probed, and storage is defined to be the size of T (total number of cells). Yao [4] shows that storage $n + 1$ and worst case query time 2 can be simultaneously attained provided that m grows at least exponentially in n ($m \geq e^{2n}$ suffices). Yao and Tarjan [3] show that $O(n)$ storage and worst case query time $O(\log m/\log n)$ can generally be attained. Therefore, if m is polynomially bounded in n, or grows at least exponentially in n, then linear storage and constant query time can be simultaneously achieved. However, as Yao [4] points out, there is an intermediate range for m, for example, $m = 2^{n^{1/2}}$, for which the possibility of linear storage and constant query time is not settled by the results quoted above.

In the next section we describe a storage technique achieving linear storage and constant worst case query time for all m and n. The query algorithm is especially easy to implement and the relative magnitudes of m and n play no role in the proofs. Section 3 discusses a general framework that motivates our construction. In Section 4 we describe a refinement that attains space $n + o(n)$, while retaining constant query time. Section 5 describes some variations of our method.

2. *Basic Representation and Query Algorithm*

In this section we illustrate the main idea behind our set representation method with a technique achieving linear storage and constant query time. Let $U = \{1, \ldots, m\}$. To simplify the discussion we assume that $p = m + 1$ is a prime number. We use the notation $a \bmod b$ to denote the unique integer x, $1 \leq x \leq b$, such that $x \equiv a (\bmod\ b)$. We need the following lemma.

LEMMA 1. *Given $W \subseteq U$ with $|W| = r$, and given $k \in U$ and $s \geq r$, let $B(s, W, k, j) = |\{x \mid x \in W \text{ and } (kx \bmod p) \bmod s = j\}|$ for $1 \leq j \leq s$. In words, $B(s, W, k, j)$ is the number of times the value j is attained by the function $x \to (kx \bmod p) \bmod s$ when x is restricted to W. Then there exists a $k \in U$ such that*

$$\sum_{j=1}^{s} \binom{B(s, W, k, j)}{2} < \frac{r^2}{s}.$$

PROOF. We show that

$$\sum_{k=1}^{p-1} \sum_{j=1}^{s} \binom{B(s, W, k, j)}{2} < \frac{(p-1)r^2}{s} \tag{1}$$

from which the Lemma follows immediately. The sum in (1) is the number of pairs $(k, \{x, y\})$, with $x, y \in W$, $x \neq y$, $1 \leq k < p$, such that

$$(kx \bmod p) \bmod s = (ky \bmod p) \bmod s.$$

The contribution of $\{x, y\}$, $x \neq y$, to this quantity is at most the number of k such that

$$k(x - y) \bmod p \in \{s, 2s, 3s, \ldots, p - s, p - 2s, p - 3s, \ldots\}. \tag{2}$$

Because $x - y$ has a multiplicative inverse mod p, the number of k satisfying (2) is $\leq 2(p - 1)/s$. Summing over the $\binom{r}{2}$ possible choices for $\{x, y\}$, we conclude that the sum in (1) is indeed bounded by $(p - 1)r^2/s$, completing the proof. □

COROLLARY 1. *There exists a $k \in U$ such that $\sum_{j=1}^{r} B(r, W, k, j)^2 < 3r$.*

PROOF. Combine Lemma 1 with the observation that $\sum_{j=1}^{r} B(r, W, k, j) = |W| = r$. □

COROLLARY 2. *There exists a $k' \in U$, such that the mapping $x \to (k'x \bmod p) \bmod r^2$ is one-to-one when restricted to W.*

PROOF. Choosing $s = r^2$, Lemma 1 provides a k' such that $B(r^2, W, k', j) \le 1$ for all j. □

Given $S \subseteq U$, $|S| = n$, our technique for representing the set S works as follows. The content k of cell $T[0]$ is used to partition S into n blocks W_j, $1 \le j \le n$, as determined by the value of the function $f(x) = (kx \bmod p) \bmod n$; pointers to corresponding blocks T_j in the memory T are provided in locations $T[j]$, $1 \le j \le n$. More specifically, a k is chosen satisfying Corollary 1 (with $W = S$ and $r = n$), so that $\sum |W_j|^2 < 3n$. The amount of space allocated to the block T_j for W_j is $|W_j|^2 + 2$. The subset W_j is resolved within this space by using the perfect hash function provided by Corollary 2 (setting $W = W_j$ and $r = |W_j|$). In the first location of T_j we store $|W_j|$, and in the second location we store the value k' provided by Corollary 2; each $x \in W_j$ is stored in location $[(k'x \bmod p) \bmod |W_j|^2] + 2$ of block T_j.

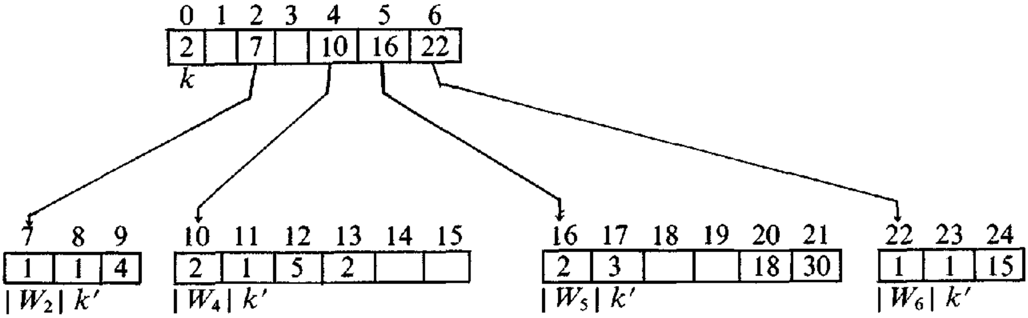
A membership query for q is executed as follows:

1. Set $k = T[0]$ and set $j = (kq \bmod p) \bmod n$.
2. Access in $T[j]$ the pointer to block T_j of T and use this pointer to access the quantities $|W_j|$ and k' in the first two locations of block T_j.
3. Access cell $((k'q \bmod p) \bmod |W_j|^2) + 2$ of block T_j; q is in S if and only if q lies in this cell.

A query requires five probes, and our choice of k in Corollary 1 implies that the size of T is at most $6n$. An example is provided below.

Example

$$m = 30, \quad p = 31, \quad n = 6, \quad S = \{2, 4, 5, 15, 18, 30\}$$



A query for 30 is processed as follows:

1. $k = T[0] = 2$, $j = (30 \cdot 2 \bmod 31) \bmod 6 = 5$.
2. $T[5] = 16$, and from cells $T[16]$ and $T[17]$ we learn that block 5 has two elements and that $k' = 3$.
3. $(30\,k' \bmod 31) \bmod 2^2 = 4$. Hence, we check the $4 + 2 = 6$th cell of block 5 and find that 30 is indeed present.

The time required to construct the representation for S might be as bad as $O(mn)$ in the worst case; finding k may require testing many possible values before a suitable one is found. However, by increasing the size of T by a constant factor, we can show that the representation can be constructed in random expected time $O(n)$, independently of m and S. Namely, we use the following variants of Corollaries 1 and 2.

COROLLARY 3. *For at least one-half of the values k in U,*

$$\sum_{j=1}^{r} B(r, W, k, j)^2 < 5r.$$

PROOF. We use eq. (1) and the fact that at most one-half of the terms in a sequence can exceed twice the average value of the sequence to conclude that

$$\sum_{j=1}^{r} \binom{B(r, W, k, j)}{2} < 2r$$

for at least one-half of the values k in U, from which the corollary follows easily. □

COROLLARY 4. *The mapping* $x \rightarrow (k'x \bmod p) \bmod 2r^2$ *is one-to-one when restricted to W for at least one-half of the values k′ in U.*

PROOF. We set $s = 2r^2$ in eq. (1) and conclude that

$$\sum_{j=1}^{2r^2} \binom{B(2r^2, W, k', j)}{2} < 1$$

for at least one-half of the values k' in U, which implies the corollary. □

Using Corollaries 3 and 4, we represent a set S of size n as before, except that now we allocate space $2\,|W_j|^2 + 2$ in storing a block W_j of S. What we gain is the fact that the probability that a particular choice for k (or k') is suitable, exceeds $\frac{1}{2}$. The choices for k (or k') are selected at random until suitable values are found.

By modifying our methods slightly, we can guarantee a worst case construction time of $O(n^3 \log m)$.

LEMMA 2. *There exists a prime* $q < n^2 \log m$ *that does not divide any of the elements in S, and that separates these elements into distinct residue classes mod q.*

PROOF. For $S = \{x_1, \ldots, x_n\}$ let $t = \prod_{i<j} (x_i - x_j) \prod_i x_i$. Clearly, $\log |t| \le \binom{n+1}{2} \log m$. Since the prime number theorem gives $\log(\prod_{q<x,\, q \text{ prime}} q) = x + o(x)$, we conclude that some prime $q < n^2 \log m$ cannot divide t. This prime q satisfies the lemma. □

We proceed as follows. If $m < n^2 \log n$, then $O(nm) = O(n^3 \log m)$. If $m \ge n^2 \log n$, then in time $O(nq)$ we produce a prime q satisfying Lemma 2 and store it in location $T[-1]$. The remainder of T is specified as before, except that the location where $x \in S$ gets stored is determined by using the hash value $x \bmod q$ in place of x, in effect replacing U with a smaller universe, $U' = \{1, \ldots, q - 1\}$, with $q \le n^2 \log m$. The total construction time is bounded by $O(nq) = O(n^3 \log m)$.

3. *Discussion*

The scheme described above can be couched in the following general framework. The value k in $T[0]$ induces a coloring of U with n colors, namely $x \rightarrow (kx \bmod p) \bmod n$. Yao's two-probe method is likewise based on an indexed family $\chi = \{C_k\}$, $|\chi| \le m$, of n-colorings having the property that for each $S \subseteq U$, $|S| =$

n, there exists a C_k in χ that is one-to-one when restricted to S. Yao refers to such a family χ as a separating system. With Yao's method, the set S is stored in T by placing k in $T[0]$, to invoke the coloring C_k, and placing $x \in S$ in $T[j]$ where j is the color of x under C_k. This approach works provided that such a χ exists. The restriction $|\chi| \leq m$ arises from the fact that its elements are indexed by the permissible range of $T[0]$. A simple counting argument shows that at least $\binom{m}{n}/(m/n)^n$ colorings are required for a separating system, from which we deduce that $m \gtrsim n^n/n!$. R. Graham uses a probabilistic argument to show that if $m \gtrsim n^{n+2}/n! \approx e^n$ then a separating system χ exists.

To extend Yao's method when $m = \exp(o(n))$, we resign ourselves to the fact that collisions are inevitable under the coloring induced by $T[0]$. Referring to the monochromatic blocks of S as bins, we attempt to use secondary colorings to separate the elements within bins. If a bin size b is sufficiently small; that is, $b \leq \log m$, then that bin can be resolved by choosing a b-coloring from a family χ' that comprises a separating system for subsets of size b.

Now a probabilistic argument shows that for all $m \gtrsim n$, there exists a family of n-colorings χ, $|\chi| \leq m$, such that for each $S \subseteq U$, $|S| = n$, there exists a coloring $C \in \chi$ that partitions S into bins of size $< \log n \leq \log m$. Therefore, we conclude from this reasoning that there exist table storage schemes under Yao's model with $O(1)$ query time and $O(n)$ storage. However, we have not been able to explicitly construct a class of storage schemes for all $m \geq n$ along these lines. We refer to storage schemes of this kind, where bin sizes are uniformly bounded by $\log m$, as L^∞ schemes.

Returning again to Yao's two-probe method, we consider the possibility of utilizing more table space, in effect using t-colorings with $t \geq n$ to completely resolve the elements of an n set S. Again, using counting and probabilistic arguments, we can show that a family χ of t-colorings exists, $|\chi| \leq m$, that resolves all S of size n, provided that m is roughly at least $\exp(n^2/t)$, which is roughly best possible. Therefore, by choosing $t = n^2$, we remove any constraint on m.

Although using n^2 colors, or equivalently space n^2 is very inefficient in terms of our original problem, it is reasonable to use b^2 colors to resolve bins of size b, provided that $\sum b^2$ is small. Probabilistic arguments show, in fact, that almost all families of n-colorings χ with $|\chi| = m$ achieve $\sum b^2 = O(n)$ for every S of size n, for all $m \gtrsim n$. This provides another class of linear space, constant query time table storage schemes, which we refer to as L^2 schemes. Contrary to the difficulty we have in constructing explicit L^∞ schemes, the construction in Section 2 provides an explicit class of L^2 schemes.

4. *Refinement*

In this section we show how to reduce the storage utilization to $n + o(n)$ while retaining constant query time. First, we provide a sketch. Our data structure in Section 2 involves an initial partition of S into n blocks, followed by resolutions of these blocks at the second level of the data structure. Our refinement involves an initial partition of S into a larger number of blocks, $g(n)$ (to be specified below), of which, obviously, at most n are nonempty. Those blocks that have more than one element are resolved at the second level as before. However, there will be very few blocks with more than one element; and moreover, the total space required to resolve them is only $o(n)$. The element of a singleton block is directly stored in the initial level of the data structure. To reduce the space requirement for the initial level of the data structure from $g(n)$ to $n + o(n)$, we use an auxiliary data structure (to be described).

Choosing $W = S$, $s = g(n)$, and $r = n$ in Lemma 1, we find that for some $k \in U$,

$$\sum_{j=1}^{g(n)} \binom{B(g(n), S, k, j)}{2} = O\left(\frac{n^2}{g(n)}\right). \tag{3}$$

Since $x^2 = O(\binom{x}{2})$ for $x \geq 2$, eq. (3) implies that

$$\sum{}' \; B(g(n), S, k, j)^2 = O\left(\frac{n^2}{g(n)}\right) \tag{4}$$

where $\sum'$ denotes the sum over all j such that $1 \leq j \leq g(n)$ and $B(g(n), S, k, j) \geq 2$. The set S is partitioned into blocks as determined by the values of the function $f(x) = (kx \bmod p) \bmod g(n)$. Since $g(n)$ will be chosen so that $\lim n/g(n) = 0$, eq. (4) implies that the total space required to resolve those blocks having two or more elements (using the method in Section 2) is $o(n)$.

In processing a membership query for q, we first determine the number $j = (kq \bmod p) \bmod g(n)$ of the block W_j of the partition of S to which q must belong if q belongs to S. At most n of these blocks are nonempty. With each nonempty block W_j we associate a cell of T in which we store either (a) the single item of W_j in the event that $|W_j| = 1$, or (b) a pointer to the second level of our data structure where W_j is resolved if $|W_j| \geq 2$. We also use a tag bit to indicate which of (a) or (b) applies. (These tag bits can be packed into $O(n/\log m) = o(n)$ words.) This approach requires an auxiliary data structure to determine whether a block W_j is nonempty, and to find the cell and tag bit associated with W_j when W_j is nonempty. The design of this auxiliary data structure is a slight modification of a similar construction due to Tarjan and Yao [3]. The cells associated with nonempty W_j are arranged consecutively with increasing j. Let T' designate the portion of T in which these cells are located. We partition the interval $I = [1, g(n)]$ into $n^2/g(n)$ subintervals of size $(g(n)/n)^2$. With each of the $n^2/g(n)$ subintervals σ of I, we associate a base address $B[\sigma]$, which is the address of the location immediately preceding the cells in T' associated with nonempty W_j, $j \in \sigma$. These base addresses are stored in a table of size $n^2/g(n) = o(n)$. A second table $A[j]$, $j \in I$, is used to store offsets: $A[j] = 0$ if $W_j = \phi$, otherwise $B[\sigma] + A[j]$ is the address in T' associated with W_j for $j \in \sigma$. Since $A[j]$ assumes at most $(g(n)/n)^2 + 1$ possible values, the entire table $A[j]$, $j \in I$ can be packed into $O(g(n)\log(g(n)/n)/\log n)$ cells of T. Picking $g(n) = n(\log n)^{1/2}$ the resulting space requirement for the $A[j]$ table is $o(n)$, and so the total space requirement for our data structure is $n + o(n)$.

The remarks at the end of Section 2 concerning the time required to construct the representation for S carry over and apply here.

5. *Variations*

The results presented here remain valid if we substitute the mapping $x \to \lfloor (kx \bmod p) \cdot s/p \rfloor$ in place of $(kx \bmod p) \bmod s$. Presumably, many other suitable mappings can be found. Another mapping that may be of interest, particularly if the multiplication of large numbers is considered objectionable, is the following. Assume that U is the set of d dimensional s-ary vectors where s is a prime. Given two vectors $\mathbf{k} = (k_1, \ldots, k_d)$ and $\mathbf{x} = (x_1, \ldots, x_d)$ in U, we let $\mathbf{k} \cdot \mathbf{x}$ denote the inner product: $\mathbf{k} \cdot \mathbf{x} = \sum k_i x_i \bmod s$. Then the analog to Lemma 1 holds for the mapping $\mathbf{x} \to \mathbf{k} \cdot \mathbf{x}$. This mapping avoids multiplication by large numbers and has the further advantage that $\mathbf{k} \cdot \mathbf{x}$ can be computed more rapidly for "short" $\mathbf{x}$ ($\mathbf{x}$ with small Hamming weight). Our data structure, however, requires a variety of such mappings

(s is a bounded parameter), which in turn requires that it be easy to convert between different representations (having differing values of s) of the elements in U. We would also like the Hamming weight to be roughly preserved in switching between representations. An obvious way to accomplish this is to use a block code approach to these representations, of which the binary coded decimal is an example.

REFERENCES

1. JAESCHKE, G. Reciprocal hashing: A method for generating minimal perfect hashing functions. *Commun. ACM 24*, 12 (Dec. 1981), 829–833.
2. SPRUGNOLI, R. Perfect hashing functions: A single probe retrieving method for static sets. *Commun. ACM 20*, 11 (Nov. 1977), 841–850.
3. TARJAN, R. E., AND YAO, A. C.-C. Storing a sparse table. *Commun. ACM 21*, 11 (Nov. 1979), 606–611.
4. YAO, A. C.-C. Should tables be sorted? *J. ACM 28*, 3 (July 1981), 615–628.

RECEIVED DECEMBER 1982; REVISED JANUARY 1984; ACCEPTED FEBRUARY 1984