Succinct Indexable Dictionaries with Applications to Encoding k-ary Trees and Multisets *

Rajeev Raman† Venkatesh Raman‡ S. Srinivasa Rao§

Abstract

We consider the *indexable dictionary* problem which consists in storing a set $S \subseteq \{0, \ldots, m-1\}$ for some integer m, while supporting the operations of *rank*(x), which returns the number of elements in S that are less than x if $x \in S$, and -1 otherwise; and *select*(i) which returns the i-th smallest element in S.

We give a structure that supports both operations in $O(1)$ time on the RAM model and requires $\mathcal{B}(n,m) + o(n) + O(\lg\lg m)$ bits to store a set of size n, where $\mathcal{B}(n,m) = \lceil \lg \binom{m}{n} \rceil$ is the minimum number of bits required to store any n-element subset from a universe of size m. Previous dictionaries taking this space only supported (yes/no) membership queries in $O(1)$ time. In the cell probe model we can remove the $O(\lg\lg m)$ additive term in the space bound, answering a question raised by Fich and Miltersen, and Pagh.

We also present two applications of our dictionary structure:

- An information-theoretically optimal representation for *k-ary cardinal trees* (aka k-ary tries). Our structure uses $\mathcal{C}(n,k) + o(n + \lg k)$ bits to store a k-ary tree with n nodes and can support parent, i-th child, child labeled i, and the degree of a node in constant time, where $\mathcal{C}(n,k)$ is the minimum number of bits to store any n-node k-ary tree. Previous space efficient representations for cardinal k-ary trees required $\mathcal{C}(n,k) + \Omega(n)$ bits.
- An optimal representation for multisets where (appropriate generalisations of) the *select* and *rank* operations can be supported in $O(1)$ time. Our structure uses $\mathcal{B}(n, m+n) + o(n) + O(\lg\lg m)$ bits to represent a multiset of size n from an m element set; the first term is the minimum number of bits required to represent such a multiset.

*Research supported by UK-India Science and Technology Research Fund project number 2001.04/IT and EPSRC grant GR L/92150

†Department of Maths and Computer Science, University of Leicester, Leicester LE1 7RH, UK. r.raman@mcs.le.ac.uk.

‡Institute of Mathematical Sciences, Chennai, India 600 113, vraman@imsc.ernet.in

§Institute of Mathematical Sciences, Chennai, India 600 113, ssrao@imsc.ernet.in

1 Introduction

Given a set $S \subseteq U = \{0, \ldots, m-1\}$, $|S| = n$ we consider the problem of representing S to support the following operations in constant time:

rank(x, S)**:** Given $x \in U$, return -1 if $x \notin S$ and $|\{y \in S | y < x\}|$ otherwise, and

select(i, S)**:** Given $i \in \{1, \ldots, n\}$, return the i-th smallest element in S.

When there is no confusion, we will sometimes omit the set S from the description of these operations. We call this the *indexable dictionary* problem, and a representation for this problem where both these operations can be supported in constant time, the *indexable dictionary representation*. This is a generalisation of the classical dictionary problem, where we need to support (yes/no) membership queries [6, 12, 17, 19], and can be thought of as combining the functionalities of the hash table and sorted array representations of S. A variant of this problem was considered previously by Elias [10].

We assume a unit-cost RAM model in which standard arithmetic and bitwise boolean operations on word-sized operands take constant time, where the word size is $\Theta(\lg m)$ bits [2, 13]. Our interest is in highly space-efficient or *succinct* representations that support both operations in $O(1)$ time.

Since there are $\binom{m}{n}$ subsets of size n, the lower bound on space for this problem is $\mathcal{B}(n,m) = \lceil \lg \binom{m}{n} \rceil = n \lg(me/n) - \Theta(n^2/m) - O(\lg n)$, as n grows and $m \geq n$. For the classical dictionary problem, Pagh [17] showed that $O(1)$-time membership queries can be supported using $\mathcal{B}(n,m) + o(n) + O(\lg\lg m)$ bits of space, improving Brodnik and Munro's previous result [6]. Raman and Rao [18] considered *dictionaries with rank*, which support $O(1)$-time *rank*() queries and gave a representation requiring $n\lceil \lg m \rceil + O(\lg\lg m)$ bits of space; this is better than augmenting Pagh's data structure with $n\lceil \lg n \rceil$ bits of explicit rank information. Raman and Rao's data structure can support *select*() queries using $n(\lceil \lg m \rceil + \lceil \lg n \rceil) + O(\lg\lg m)$ bits, but this is nearly $2n \lg n$ bits more than necessary.

In this paper we give a solution that requires $\mathcal{B}(n,m) + o(n) + O(\lg\lg m)$ bits, which is space-optimal

to within lower-order terms. In the *cell probe* model with word size $\Theta(\lg m)$, a variant of this data structure requires $\mathcal{B}(n,m)+o(n)$ bits. The latter result implies that n words of $\lceil \lg m \rceil$ bits suffice to answer membership queries in $O(1)$ time on a set of size n. This answers a question raised by Fich and Miltersen [11] and Pagh [17]. By contrast, Yao's influential paper [21] showed that if the n words must contain a permutation of S, then membership queries cannot be answered in $O(1)$ time using n words.

The main ingredient in our results is an idea we call *most-significant-bit first (MSB) bucketing*. The idea is to apply a trivial top-level hash function to the keys in S, which simply takes the value of the t most significant bits of a key. As we can omit the t most significant bits of all keys that "hash" to the same bucket, space savings is possible. A similar idea was used by Brodnik and Munro[6] in their succinct representation of sets. A major difference between our approach and theirs is that they store explicit pointers to refer to the representation of buckets, which uses more space than necessary (and hence constrains the number of buckets). Instead, we use a succinct representation of the multiset of bucket indices that not only provides the extra functionality needed for supporting *rank* and *select*, but also uses significantly less space.

Another related technique of *quotienting* [17] stores only the "quotients" of keys that are mapped to a bucket by a standard hash function (e.g. those of [12]). The crucial difference is that MSB bucketing preserves enough information about ordering of keys to allow us to maintain most of the rank information using negligible extra space. In addition we use ideas of range reduction ([12] and others) and distinguishing bits ([1] and others) to achieve our main result.

In addition to being fundamental problems in their own right, these dictionaries also find use in succinct representations of *cardinal* trees (or tries) and multisets. A k-ary cardinal tree is a rooted tree, each node of which has k positions labeled $0,\ldots,k-1$, which can contain edges to children. The space lower bound for representing a k-ary cardinal tree with n nodes is $\mathcal{C}(n,k) = \left\lceil \lg \left(\frac{1}{kn+1}\binom{kn+1}{n} \right) \right\rceil$; it can be seen that $\mathcal{C}(n,k) \approx (\lg(k-1) + k \lg \frac{k}{k-1})n$ for constant k and growing n, and that $\mathcal{C}(n,k) = (\lg k + \lg e)n - o(n + \lg k)$ bits if k grows with n. We are interested in finding succinct representations of cardinal k-ary trees that support navigation and other queries in $O(1)$ time.

Benoit et al. [4] gave a cardinal tree structure that takes $(\lceil \lg k \rceil + 2)n + o(n) = \mathcal{C}(n,k) + \Omega(n)$ bits of space that answers queries asking for parent, ith child, degree and subtree size in constant time and queries asking for a child with label i in $O(\lg \lg k)$ time in the worst case. The time for the latter query was improved to $O(\lg \lg \lg k)$ in [18] and to $O(1)$ in [5], with a slight increase in space to $(\lceil \lg k \rceil + 2)n + o(n) + O(\lg \lg k)$ bits.

Using our indexable dictionary, we obtain an optimal encoding for k-ary cardinal trees taking $\mathcal{C}(n,k) + o(n) + O(\lg \lg k)$ bits, in which all the above operations, except the subtree size at a node, can be supported in constant time. Our representation uses a numbering of tree nodes that allows us to dispense with an explicit encoding of the tree structure, a feature shared with the representation of [9]. All the above results on cardinal trees use the unit-cost RAM model with a word size of $\Theta(\lg(\max\{n,k\}))$ bits.

We also observe that our indexable dictionary structure can be applied to obtain an information-theoretically optimal encoding for *indexable multisets*, which support operations *rankm* and *selectm* that are natural generalisations of *rank* and *select* to multisets. Let M be a multiset of n numbers from $\{0,\ldots,m-1\}$. It is easy to see that $\mathcal{B}(n,m+n)$ is a lower bound on the number of bits needed to represent such a multiset, as there is a $1-1$ mapping between such multisets and sets of n elements from $\{0,\ldots,m+n-1\}$ (see e.g. [10]). However, if we transform a multiset into a set by this mapping, then *rankm* and *selectm* do not appear to translate into *rank* and *select* operations on the underlying set. Using some additional ideas, we obtain an multiset representation that takes $B(n,m+n) + o(n) + O(\lg \lg m)$ bits.

The next section gives some preliminary results including optimal representations for indexable dictionaries and multisets when n is sufficiently large with respect to m. In Section 3 we give our general optimal representation of an indexable static dictionary. Section 4 deals with the application to indexable multisets. Section 5 deals with the representation of k-ary trees. In Section 6, we give an optimal space indexable dictionary in the cell probe model.

2 Preliminaries

We state here some lemmas with references or sketches for proofs, which will be used later in the paper. In what follows, if f is a function defined from a finite set X to a finite totally ordered set Y, by $||f||$, we mean $\max\{f(x) : x \in X\}$. We use the notation $[m]$ to denote the set $\{0,1,\ldots,m-1\}$.

Given a set $S \subseteq U$, a *fully indexable dictionary (FID)* representation for S supports *rank*() and *select*() operations on both S and its complement $\bar{S} = U \setminus S$ in $O(1)$ time. A FID is quite a powerful data structure for finite universes. For example, it is easy to check that if $U = [m]$ then a FID representation for S supports

$O(1)$-time predecessor queries[1] on S. It is known that in general, predecessor queries cannot be answered in $O(1)$ time on the RAM model by a data structure that uses $n^{O(1)}$ words of space [3]. Hence FIDs cannot exist in general unless they are relatively space-inefficient. However, we show that FIDs that require essentially $\mathcal{B}(n,m)$ bits exist for sufficiently dense sets, generalising a result of Pagh [17]. FIDs are essential to our data structure as they are intimately related to operations on multisets, as we note in the next section and in Theorem 4.1.

2.1 Multisets. Given a multiset M of n numbers from the set $U = [m]$, define the operations:

rankm(x)**:** Given $x \in U$, return -1 if $x \notin M$ and $|\{y \in M | y < x\}|$ otherwise,

selectm(i)**:** Given $i \in \{1, \ldots, n\}$, return the largest element $x \in M$ such that *rankm*$^+(x) \leq i-1$ and

rankm$^+(x)$**:** Given $x \in U$, return $|\{y \in M | y < x\}|$.

We now make the connection between FIDs and multiset representations that support *rankm*$^+$ and *selectm*:

LEMMA 2.1. *Suppose there is a fully indexable dictionary representation for any given set $S \subseteq U$ using $f(|S|, |U|)$ bits of space. Then given a multiset M of n numbers from the set $[m]$, there is a structure to represent M using $f(n, m+n)$ bits of space that supports rankm$^+$ and selectm in constant time.*

Proof. Consider the following bit representation [10] of the multiset M: For $i = 0$ to $m-1$ in that order, represent i by a 0 followed by n_i 1's where n_i is the number of times the element i appears in the multiset M. Clearly this representation takes $m+n$ bits since it has m 0s and n 1s.

Now view this bit sequence as a characteristic vector of a set S of n elements from the universe $[m+n]$. Represent S as a FID using $f(n, m+n)$ bits. It is easy to verify that *rankm*$^+(x) =$ *select*$(x+1, \bar{S}) - x$ and *selectm*$(i) =$ *select*$(i, S) - i$. □

2.2 Fully indexable dictionaries. Given a bit vector of length n one can augment it with $o(n)$ additional bits of information so as to support the following operations in $O(1)$ time [14, 15] (see also [16]): (i) given i, report the number of 0s (1s) in positions $1, \ldots, i-1$ of the bit vector; (ii) given i, return the index of the i-th 0 (1) in the bit vector. Applying this to a set represented explicitly as its characteristic vector, we get:

LEMMA 2.2. ([14, 15] (SEE ALSO [16])) *Given a set $S \subseteq [m]$ there is a FID on S that requires $m + o(m)$ bits.*

Lemmas 2.2 and 2.1 give:

COROLLARY 2.1. *A multiset M of n numbers from the set $[m]$ can be represented using $m+n+o(m+n)$ bits of space to support rankm$^+$ and selectm in constant time.*

We now consider more space-efficient FIDs, and show the following lemma which is an extension of [17, Proposition 4.3].

LEMMA 2.3. *Given a set $S \subseteq [m]$, $|S| = n$, there is a FID on S that requires $\mathcal{B}(n,m) + O(m \lg\lg m / \lg m)$ bits of space.*

Proof. Let $U = [m]$ be the universe. We divide the universe into blocks of $u = \frac{1}{2}\lfloor \lg m \rfloor$ numbers each, with $U_i = \{iu, \ldots, (i+1)u - 1\}$ for $1 \leq i \leq 2m/\lg m$. We store two arrays of size $O(m/\lg m)$. Let $S_i = S \cap U_i$ and $n_i = |S_i|$. The first array A stores the numbers n_i in equal sized fields of $\lceil \lg u \rceil$ bits each. The second array B stores the quantities $\lceil \mathcal{B}(n_i, u) \rceil$; since $\mathcal{B}(n_i, u) \leq u$ these numbers can also be stored in equal sized fields of $\lceil \lg u \rceil$ bits each. This requires $O(m \lg\lg m / \lg m)$ bits of space. We also store the prefix sums of the two arrays as described in [17, Proposition 4.2] or [20] in $O(m \lg\lg m / \lg m)$ bits again such that the i-th prefix sum is calculated in $O(1)$ time.

The set S_i is represented by a string of $\lceil \mathcal{B}(n_i, u) \rceil$ bits by storing an index into a table containing the characteristic bit vectors of all possible subsets of size n_i from a universe of size u, and S is represented by concatenating the representations of S_i's. The length of this representation is at most $\mathcal{B}(n,m)$ bits, as shown in Lemma 4.1 of [17]. We also store precomputed tables to support *rank* and *select* queries on the sets S_i given its size and the representation. These tables require $O(m^{1-\epsilon})$ bits of space for some fixed positive constant $\epsilon < 1$.

To support *select* we do the following. Let $p = |A| (= 2m/\lg m)$. We explicitly store the positions in A of *select*$(i\ (\lg p)^2)$ for $1 \leq i \leq p/(\lg p)^2$. Here *select*$(j)$ gives the position l in A such that $\Sigma_{i=1}^{l-1} A[i] < j \leq \Sigma_{i=1}^{l} A[i]$. Call the subset of the elements of the universe between two successive positions as a block. Following the ideas of [7, 16], we call a block 'dense' if the block size is at most $(\lg p)^4$ and 'sparse' otherwise. Thus a sparse block has size $> (\lg p)^4$ but the sum of the numbers in the block is roughly $(\lg p)^2$. We represent a

[1] The predecessor of x in S for any $x \in U$ is defined as $\max\{y \in S | y < x\}$.

sparse block by explicitly listing the positions *select*(j) with respect to the beginning of the block, for $1 \leq j \leq (\lg p)^2$ in the block. This takes at most $(\lg p)^3$ bits for each sparse block. Thus the total space used for all sparse blocks together is at most $O(m(\lg p)^3/(\lg p)^4)$ which is $O(m/\lg m)$ bits.

For a dense group, we construct a complete tree with branching factor $\sqrt{\lg p}$ with the block elements at the leaves. At each node of this tree, except at the leaves, we store an array containing the sum of all the numbers in each of its child subtrees. This requires $O(k \lg\lg p/\lg p)$ bits for a dense block of size k (we do not count the space for storing the arrays at the level above the leaf nodes as these are part of array A). Thus the total space required for all the dense blocks together is at most $O(m \lg\lg m/\lg m)$ bits.

To compute *select*(i) we find the predecessor of i in the partial sum structure for the array A. This gives us the block to which the ith smallest element of S belongs to. If that block is sparse, we know from its representation, the position of the required element (as these pointers are stored explicitly). Otherwise, if the block is dense, we start at the root of the tree structure corresponding to that block and do a predecessor search among the numbers stored in the array stored at that node to find the subtree to which the required element belongs. This can be done in constant time using word-level parallelism, as the array at each node takes $O(\sqrt{\lg p}\lg\lg p) = o(\lg n)$ bits. We do this repeatedly at each level looking for an appropriately ranked element, going down the tree (spending constant time at each level) until we reach a leaf. This leaf corresponds to some subset S_j. We find the number of elements k in $S_0 \cup \ldots \cup S_{j-1}$ using the partial sum structure for the array A, index into the string for S to get the representation of S_j using the partial sum structure for the array B and find the position l of the $(i-k)$th element in the representation of S_i using a table lookup. Then it is easy to see that *select*$(i) = l + ku$. Since the tree is of constant depth, it follows that we can find *select*(i) in constant time.

To find *rank*(x) we proceed as in [17]: first compute $i = \lfloor x/u \rfloor$, find the number of elements in $S_0 \cup \ldots \cup S_{i-1}$ using the partial sum structure for the array A, index into the string for S to get the representation of S_i using the partial sum structure for the array B and find the rank of x within the set S_i using a table lookup.

For supporting *rank* and *select* operations on $\bar{S}$, we need to store the arrays A and B corresponding to the set $\bar{S}$ and their 'partial sum' and 'select' structures. We need not store the representations of $\bar{S}_i = U_i \setminus S_i$ separately, as we can use the representation of S_i itself for this purpose (as there is a one-to-one mapping between them). The *rank* and *select* operations on $\bar{S}_i$, given its 'representation', can be supported in constant time using lookup tables. These tables require $O(m^{1-\epsilon})$ bits of space for some fixed positive constant $\epsilon < 1$.

The space required for the whole structure is clearly $\mathcal{B}(n,m) + O(m \lg\lg m/\lg m)$ bits. □

It immediately follows that:

COROLLARY 2.2. *There is a fully indexable dictionary representation for a set $S \subseteq [m]$, $|S| = n$ that uses $\mathcal{B}(n,m) + o(n)$ bits of space, provided that m is $O(n\sqrt{\lg n})$.*

The following corollary follows from Corollary 2.2 and Lemma 2.1. Note that $\mathcal{B}(n, m+n)$ is the information theoretic minimum number of bits to represent a multiset on n elements from an m element universe.

COROLLARY 2.3. *If m is $O(n\sqrt{\lg n})$, then a multiset M of n numbers from the set $[m]$ can be represented using $\mathcal{B}(n, m+n) + o(n)$ bits to support rankm$^+$ and selectm in constant time.*

2.3 A Simple Indexable Dictionary. The following lemma is a simplification and extension of one from [5]:

LEMMA 2.4. *Let $M, N \geq 1$ be integers, let $S \subseteq [M]$ be a set of size $n \leq N$. Suppose we have access to two functions $h_S(x)$ and $q_S(x)$, defined on $[M]$, satisfying the following conditions:*

1. *h_S is 1-1 on S.*
2. *h_S and q_S can be evaluated in $O(1)$ time and from $h_S(x)$ and $q_S(x)$ one can uniquely reconstruct x in $O(1)$ time.*
3. *$||h_S||$ is $O(N^2)$ if $n > \sqrt{\lg N}$ and $||h_S||$ is $(\lg N)^{O(1)}$ otherwise.*
4. *$\lceil \lg ||h_S|| \rceil + \lceil \lg ||q_S|| \rceil$ is $\lg M + O(1)$.*

Then we can represent S using $n \lg M + n \lg n + O(n)$ bits and support rank and select in $O(1)$ time, assuming a word size of $\geq \lg \max\{M, N\}$ bits and a precomputed table of size $o(N)$ bits.

Proof. Let $x_1 < x_2 < \ldots < x_n$ be the elements of S. If $n \leq \sqrt{\lg N}$ then write down $h_S(x_1), \ldots, h_S(x_n)$ in fields of $\lceil \lg ||h_S|| \rceil$ bits each, followed by $q_S(x_1), \ldots, q_S(x_n)$ in fields of $\lceil \lg ||q_S|| \rceil$ bits each. This requires $n \lg M + O(n)$ bits. To do *rank*(x) we calculate $h_S(x)$ and look for a match in $h_S(x_1), \ldots, h_S(x_n)$ in $O(1)$ time using table lookup (note that in this case $n \cdot \lceil \lg ||h_S|| \rceil =$

$O(n \lg\lg N) = o(\lg N)$ and hence the lookup table will be of size $o(N)$ bits). If $h_S(x) = h_S(x_i)$, then verify whether $q_S(x) = q_S(x_i)$. If so, return $i-1$, otherwise return -1. To compute *select*(i), reconstruct x_i from the values $h_S(x_i)$ and $q_S(x_i)$, which can be accessed in constant time, and return it.

If $n > \sqrt{\lg N}$, then we store the set $S' = \{h_S(x) | x \in S\}$ using the the scheme of [18] except when storing the ranks with the element $h_S(x)$ of S', store the rank of $x \in S$. As each of the rank information takes anyway only $\lceil \lg n \rceil$ bits, this structure takes $n \lceil \lg ||h_S(x)|| \rceil$ bits. We also store the sequence $q_S(x_1) q_S(x_2) \ldots q_S(x_n)$ using $n \cdot \lceil \lg ||q_S|| \rceil$ bits. So if we find $h_S(x)$ in S', for the query element x, we will also know a rank r. So we can confirm whether that value is x by comparing $q_S(x)$ with $q_S(x_r)$. In addition, we use $n \lg n + O(n)$ bits of information to support *select*() queries. □

The function h_S is essentially a 'universe reduction' function commonly used in perfect hashing. Indeed, the first two steps of the FKS scheme [12] show the existence of the function h_S with range of $O(|S|^2)$ and $q_S(x)$ is simply the quotient information required to recover x given $h_S(x)$. However, for a small set S (a set whose size is $\leq \sqrt{\lg N}$), the space to represent the function h_S can become dominant. So to reduce the total size of the function descriptions, we [5] use the same function for a group of (up to $\lg N$) small sets. This is why h_S has a relaxed range of $O((\lg N)^c)$, rather than the minimum range of $O(\lg N)$. The details are encapsulated in the lemma below, whose proof is reproduced in the appendix for completeness.

LEMMA 2.5. *[5] Let N, M be as in Lemma 2.4, and let $0 \leq i_1 < i_2 < \ldots < i_s < N$ be a sequence of integers. Let $S_{i_1}, S_{i_2}, \ldots, S_{i_s}$ all be subsets of $[M]$ such that $\sum_{j=1}^{s} |S_{i_j}| \leq N$. The there exist functions $h_{S_{i_j}}$ and $q_{S_{i_j}}$ for $j = 1, \ldots, s$ that satisfy the conditions of Lemma 2.4, and that can be represented in $o(N) + O(\lg\lg M)$ bits.*

3 Optimal Indexable Dictionary

Assuming that Lemma 2.4 is applicable to various subsets we consider, we first show that a set $S \subseteq [m]$ such that $|S| = n$ can be stored in at most $n\lceil \lg m \rceil$ bits, and can support *rank* and *select* operations in $O(1)$ time. This excludes the space for the functions needed by Lemma 2.4. Using Lemma 2.5 we represent these functions in $o(n) + O(\lg\lg m)$ bits, giving us Theorem 3.1.

Later we use the representation of Theorem 3.1 to store multiple independent (but not necessarily disjoint) dictionaries efficiently (Theorem 3.2). We then obtain our main result, Theorem 3.3.

3.1 A Sub-Optimal Solution using MSB bucketing.

THEOREM 3.1. *There is an indexable dictionary for a set $S \subseteq [m]$, $|S| = n$, that uses at most $n\lceil \lg m \rceil + o(n) + O(\lg\lg m)$ bits of space.*

Proof. The proof is as follows. We first describe a construction algorithm, in which we partition S using MSB bucketing, recursing on large partitions. The base case of the recursion is handled using Lemma 2.4. We get an overall space bound of $n\lceil \lg m \rceil$ assuming the hypothesis of Lemma 2.4 for each application of this lemma. We then show how to support *rank* and *select* in $O(1)$ time. Finally, we sketch how to use Lemma 2.5 to represent all functions used in applications of Lemma 2.4 using $o(n) + O(\lg\lg m)$ bits.

We describe the construction using a recursive function CONSTRUCT(T, r, ℓ), which takes three arguments, a set T and two integers $r, \ell \geq 0$ with $T \subseteq [2^r]$. In the description of the function, we let $t = \lceil \lg m \rceil - \lceil \lg n \rceil$, and let c and d be two constants whose values are to be determined later.

If $|S| \leq d$ then we simply write down the elements of S using $n\lceil \lg m \rceil$ bits, and we are done. Otherwise, we call CONSTRUCT with parameters $T = S$, $r = \lceil \lg m \rceil$ and $\ell = 2$. We will show later that CONSTRUCT returns a representation of S that occupies $n(\lceil \lg m \rceil - \lceil \lg n \rceil + O(1))$ bits, which is fewer than $n\lceil \lg m \rceil$ bits if d is sufficiently large. Thus the space of our representation will always be bounded by $n\lceil \lg m \rceil$ bits. The function CONSTRUCT works as follows:

CONSTRUCT(T, r, ℓ)

1. If $|T| \leq d$ then write down the elements of T explicitly, padding the output out to $|T| \cdot (t + 4\ell + c)$ bits if necessary.

2. If $\ell > 0$ then partition the elements of T according to their top $\lceil \lg |T| \rceil$ bits. This 'partitions' T into $z = 2^{\lceil \lg |T| \rceil} \leq 2|T|$ sets denoted as $T_0, \ldots, T_{z-1}$, where T_i consists of the last $r - \lceil \lg |T| \rceil$ bits of all the keys in T whose most significant $\lceil \lg |T| \rceil$ bits have value i, for $i = 0, \ldots, z-1$.

We store a representation of the multiset corresponding to the top $\lceil \lg |T| \rceil$ bits of each element of T using $|T| + z + o(T) \leq 3|T| + o(|T|)$ bits using Corollary 2.1 and pad this out to $4|T|$ bits. For $i = 0, \ldots, z-1$, we then call CONSTRUCT$(T_i, r - \lceil \lg |T| \rceil, \ell - 1)$, and concatenate the representations returned with the representation of the multiset corresponding to the top $\lceil \lg |T| \rceil$ bits of T, and return.

3. If $\ell = 0$ we apply the result of Lemma 2.4 to T, padding the output out to $|T| \cdot (t + 4\ell + c)$ bits if necessary.

We now inductively show that each of the above calls to CONSTRUCT(T, r, ℓ) returns a representation that occupies $|T| \cdot (t + 4\ell + c)$ bits when called with $\ell = 2$. First, we consider the base cases of the induction. The recursion terminates either when $|T| \leq d$ or when $|T| > d$ but $\ell = 0$.

We take the former case first, and say this occurs when $\ell = \ell^*$. Since CONSTRUCT is only called at the top level when $|S| > d$, it must be the case that $\ell^* < 2$, and that T has been created by successive partitionings at levels $2, \ldots, \ell^* + 1$, according to the most significant $r_2 \geq \ldots \geq r_{\ell^*+1}$ bits respectively. Note that $r = \lceil \lg m \rceil - r_2 - \ldots - r_{\ell^*+1} \leq \lceil \lg m \rceil - \lceil \lg n \rceil$, since $r_2 = \lceil \lg n \rceil$. Hence the values in T may be written down using $|T| \cdot r \leq |T| \cdot t$ bits, and this representation is padded out to the requisite length.

In the latter case, again T must have been created by partitioning according to the most significant $r_2 = \lceil \lg n \rceil$ bits, followed by partitioning according to $r_1 \geq \lceil \lg |T| \rceil$ bits, and so $r \leq \lceil \lg m \rceil - \lceil \lg n \rceil - \lceil \lg |T| \rceil$. The representation of Lemma 2.4 thus requires at most $|T|(r + \lg |T| + O(1))$ bits. This is no more than $|T|(t+c)$ bits, and can be padded out to this length, if c is chosen large enough.

Now suppose that a call at level $\ell > 0$ results in T being partitioned into sets $T_1, \ldots, T_z$, and that recursive calls are made on these sets at level $\ell - 1$. Inductively, the sizes of the representations returned are $|T_i|(t + 4(\ell-1) + c)$ bits, and appending them to a $4|T|$-bit representation of the partition gives a representation of T that requires $|T|(t + 4\ell + c)$ bits as required.

Therefore, at the top level, S is represented using $n(\lceil \lg m \rceil - \lceil \lg n \rceil + 8 + c) \leq n \lceil \lg m \rceil$ bits (for sufficiently large d), as required. We now show that this supports *rank* and *select* in $O(1)$ time. Again, both functions are recursive and we assume that at the start of each level of recursion, we have a pointer to the start of a representation of the set T to be searched, as well as the size of T. At the top level, $T = S$ and the claim is clearly true. Again, ℓ denotes the level of recursion, and $\ell = 2$ initially.

We now describe how the computation of *rank* proceeds; *select* works in a similar way. If $|T| < d$, where d is the constant used in CONSTRUCT, then we apply the trivial algorithm in $O(1)$ time and return. Otherwise, if $\ell = 0$, we apply Lemma 2.4 in $O(1)$ time and return. If neither of these holds, we consider the first $4|T|$ bits of the representation of T, which contains the representation of the multiset μ of the values in the top $\lceil \lg |T| \rceil$ bits of the elements of T. We extract the top $\lceil \lg |T| \rceil$ bits of the current query key[2]; suppose that these bits have value i, $0 \leq i < z = 2^{\lceil \lg |T| \rceil}$. Using Corollary 2.1, we calculate $\rho = rankm^+(i)$ and $\rho' = rankm(i+1)$ in $O(1)$ time; note that $\rho' - \rho$ is the size of the set T_i on which we will recurse. The start of the representation of T_i is also easy to compute: it starts $4|T| + \rho(t + 4(\ell-1) + c)$ bits from the start of the representation of T, where c again is the constant used in CONSTRUCT. We then remove the top $\lceil \lg |T| \rceil$ bits from the query key, decrement ℓ by one, and recurse on T_i. If the value returned by the recursive call is -1 we return -1 as well, otherwise we add the value returned by the recursive call to ρ and return. The computation is clearly constant-time.

Finally we make remark about the use of Lemma 2.5 to represent all the bottom-level sets. Implicit in Lemma 2.5 is that there is a numbering of the sets using integers from $[n]$, but we can simply take the sums of the cardinalities of the sets whose indices are less than the index of a given set. This information must be computed anyway during *rank* and *select*. This completes the proof of the Theorem 3.1. □

3.2 Representing Multiple Indexable Dictionaries.

THEOREM 3.2. *Let $S_1, S_2, \ldots, S_s$ all contained in $[m]$ be given sets with S_i containing n_i elements, such that $\sum_{i=1}^{s} n_i = n$. Then this collection of sets can be represented using $n \lceil \lg m \rceil + o(n) + O(\lg \lg m)$ bits where the rank(x, S_i) and select(j, S_i) operations can be supported in constant time for any $x \in [m], 1 \leq j \leq n$ and $1 \leq i \leq s$. We also assume that we have access to a constant time oracle which returns the starting position of the representation of each dictionary.*

Proof. If we simply apply Theorem 3.1 for each of the sets S_i's, then we get a representation taking $n \lceil \lg m \rceil + o(n) + O(s \lg \lg m)$ bits and can support the required operations in constant time. To improve the space bound to $n \lceil \lg m \rceil + o(n) + O(\lg \lg m)$ bits, we use the

[2] Standard techniques allow us to calculate $\lceil \lg x \rceil$ for sufficiently small integers x.

sharing prime idea in the proof of Lemma 2.5 where the hash functions required for small sets are shared by several of them. □

Remark: Note that $\sum_{i=1}^{j} n_i$ for $1 \leq j \leq s$, would suffice to find the starting position of the representation of S_{j+1}.

3.3 Optimal Bucketing. In this section, we develop our main result of the paper giving a representation for an indexable dictionary taking $\mathcal{B}(n,m) + o(n) + O(\lg\lg m)$ bits of space. First, if $m < 4n\sqrt{\lg n}$ then we use the "dense" result of Corollary 2.2 which establishes the result.

Otherwise, by representing the multiset containing the first $\lceil \lg n \rceil$ bits of the n elements of S using at most $3n+o(n)$ bits using Corollary 2.1 and representing the remaining bits using the structure of Theorem 3.2, it is easy to get a representation taking at most $3n + n(\lceil \lg m \rceil - \lceil \lg n \rceil) + o(n) + O(\lg\lg m)$ bits. To improve the linear term of the space bound in the above lemma to $n \lg e$ from $3n$, and to get rid of the ceilings in the second term, we first take approximately the first $\lg(n\sqrt{\lg n})$ bits of each element and represent the resulting multiset using Corollary 2.3. The details are as follows.

If $m \geq 4n\sqrt{\lg n}$, we choose an integer $l > 0$ such that $n\sqrt{\lg n} \leq \lfloor m/2^l \rfloor \leq 2n\sqrt{\lg n}$. We now group the keys based upon the mapping $b(x) = \lfloor x/2^l \rfloor$. Let $b_{\max} = \lfloor (m-1)/2^l \rfloor$. We "partition" S into sets B_i, for $i = 0, \ldots, b_{\max}$ where $B_i = \{x \bmod 2^l \mid x \in S \text{ and } b(x) = i\}$. In addition, we have the multiset $B_{top} = \{b(x) | x \in S\}$. We represent B_{top} using the structure of Corollary 2.3 taking $\mathcal{B}(n, b_{\max}+n)+o(n) = \mathcal{B}(n, m/2^l) + o(n)$ bits, which supports $rankm^+$ and *selectm* on B_{top} in constant time.

The overall representation is the following. First we represent B_{top} as above. Then we represent each of the B_i's using the construction of Theorem 3.2. The total space, in terms of the number of bits used, will be $nl + \mathcal{B}(n, m/2^l) + o(n) + O(\lg\lg m)$ which is $nl + n\lg(me/2^l n) + o(n) + O(\lg\lg m)$ which is $\mathcal{B}(n,m) + o(n) + O(\lg\lg m)$ as $m \geq 4n\sqrt{\lg n}$.

The computations of *rank* and *select* proceed essentially as in Theorem 3.1, except that we use Corollary 2.3 instead of Lemma 2.1. Thus we have the following theorem.

THEOREM 3.3. *There is an indexable dictionary for a set $S \subseteq [m]$ of size n that uses at most $B(n,m)+o(n)+O(\lg\lg m)$ bits.*

4 Applications to Multiset Representation

Given a multiset M from $[m]$, $|M| = n$, an *indexable multiset* representation for M supports the *rankm* and *selectm* operations in constant time.

In this section, we develop an optimal ($\mathcal{B}(n, m+n)$ + lower order terms) indexable multiset representation using our succinct indexable dictionary.

THEOREM 4.1. *Given a multiset M of n elements from $[m]$, there is an indexable multiset representation of M that uses $\mathcal{B}(n, m+n)+o(n)+O(\lg\lg m)$ bits and supports rankm and selectm() operations in constant time.*

Proof. If n is dense in m, i.e. if m is $O(n\sqrt{\log n})$ then the lemma follows from Corollary 2.3.

If not, then we represent the multiset M as follows. First represent the set S of distinct elements present in M using the indexable dictionary structure of theorem 3.3 using $\mathcal{B}(n', m) + o(n) + O(\lg\lg m)$ bits where $n' \leq n$ is the number of distinct elements present in M.

Then represent the rank information separately by representing each element i present in M (in increasing order) by a 1 followed by n_i 0s where n_i is the multiplicity of the element i in M. Now erase the first 0 after every 1 (since we are representing only the elements present in M, there will be at least one 0 after a 1). This representation is a bitstring of length n with n' 1's. This bitstring could be considered as a characteristic vector of a set $R \subseteq [n]$ with $|R| = n'$. Let $\bar{R} = [n] \setminus R$.

Now to find $rankm(x)$, first find $rank(x, S)$. If the answer is -1, then return -1. Otherwise $rankm(x)$ is $select(rank(x,S)+1, R)$. To find $selectm(i)$, let $r = rank(i,R)+1$ if $rank(i,R) \geq 0$ and $r = i - 1 - rank(i, \bar{R})$ otherwise. The value r is precisely the number of 1's up to and including i in the characteristic vector of R. Then $selectm(i) = select(r, S)$.

As we can see, to support both $rankm(x)$ and $selectm(i)$ in constant time, we need a fully indexable dictionary for R.

If n' is dense in n, i.e. $n = O(n'\sqrt{\lg n'})$, then use the fully indexable dictionary of Lemma 2.3 for R. This uses $\mathcal{B}(n', n) + o(n')$ bits for a total of $\mathcal{B}(n', m)+\mathcal{B}(n', n)+o(n)+O(\lg\lg m)$ including the space for representing S. It can be verified that this space is at most $\mathcal{B}(n, m+n) + o(n) + O(\lg\lg m)$.

Otherwise represent R using the FID representation of Lemma 2.2 which uses $n+o(n)$ bits. Since n' is sparse in n, $n' < cn/\sqrt{\log n}$ (for some constant c) in which case $\mathcal{B}(n', m) + n \leq \mathcal{B}(n,m) + o(n) \leq \mathcal{B}(n, m+n) + o(n)$ as n is also sparse in m. To see this, note that $\binom{n}{m} = \frac{m-n+1}{n}\binom{n-1}{m} \geq 2\binom{n-1}{m}$ since $(m-n+1)/n > 2$ for sufficiently large m and $n \leq dm/\sqrt{\lg m}$ for some constant d. Hence $\mathcal{B}(n,m) \geq \mathcal{B}(n-1, m) + 1$ and so

$\mathcal{B}(n,m) \geq \mathcal{B}(n',m) + n - n'$. That is, $\mathcal{B}(n',m) + n \leq \mathcal{B}(n,m) + n' \leq \mathcal{B}(n,m) + o(n)$. □

5 Applications for k-ary Tree Representation

In this section, we look at the representation of a k-ary cardinal tree that supports the navigational operations in constant time, using our optimal indexable dictionary. We improve the space for encoding k-ary cardinal trees by giving an encoding that takes $\mathcal{C}(n,k) + o(n) + O(\lg\lg k)$ bits of space and supports finding the parent, ith child and the child labeled j in constant time. Thus, the space for the representation is information theoretically optimal up to $o(n+\lg k)$ terms. Unfortunately, we are not able to support subtree size in this representation.

THEOREM 5.1. *A k-ary tree on n nodes can be represented using $\mathcal{C}(n,k) + o(n) + O(\lg\lg k)$ bits where given a node of the tree, we can go to its i-th child or to its child labeled j or to its parent if they exist, all in constant time. In addition, we can determine the degree of a node as well as the ordinal position of a node among its siblings in constant time.*

Proof. Consider a level-ordered labeling of the tree nodes by labels 0 to $n-1$ where in each level the labels are given in the increasing order from left to right (root gets label 0, the nodes at level 1 get the next so many labels in the increasing order from left to right etc.). Let $S = \{\langle i,j\rangle : i \in [n], j \in [k]$ and $\exists$ an edge labeled j out of the node labeled $i\}$.

Then $|S| = n-1$ and S represents the (edges of the) k-ary tree uniquely (given the labeling of the tree nodes). We map the pairs $\langle i,j\rangle$ to integers in the range $[kn]$ using the obvious mapping $\langle i,j\rangle \rightarrow i \cdot k + j$. We represent S using our representation of Theorem 3.3 using $\mathcal{B}(n-1,kn) + o(n) + O(\lg\lg kn) = \mathcal{B}(n,kn+1) - \lg(kn+1) + o(n) + O(\lg\lg kn) = \mathcal{C}(n,k) + o(n) + O(\lg\lg k)$ bits. This is our representation of the k-ary tree.

We first describe basic navigation in this tree. Let x be a label of a node in the tree, and let $y \in [k]$ be an arbitrary element.

- $rank(\langle x,y\rangle) + 1$ in S gives the label of the node pointed to by the edge $\langle x,y\rangle$, i.e. the node which is the child of the node x through the edge labeled y if there is a child labeled y for x and it returns 0 otherwise.

- The first component of $select(x)$ is the parent of the node labeled x. I.e. if the x-th element in S is $\langle i,j\rangle$, then i is the label of the parent of the node labeled x.

If we wish to support more than the above basic navigation operations, we need to be able to find the rank of $\langle x,0\rangle$ even if $\langle x,0\rangle \notin S$. We can do this by a more detailed inspection of the proof of Theorem 3.3, and potentially modifying S slightly.

If $k < 4\sqrt{\lg n}$, then the set S is dense and so this follows from Lemma 2.3. If $k > 4\sqrt{\lg n}$ then let $k' = 2^l \cdot (k \ div \ 2^l + 1)$ i.e. round the value of k to the next multiple of 2^l. We redefine S with the new value of k'; more precisely the pairs in S stay the same, but we change the mapping that takes pairs to integers as $\langle i,j\rangle \rightarrow i \cdot k' + j$.

By doing this, we ensure that no bucket at the top level of Theorem 3.3 contains elements of the form $\langle x,y\rangle$ and $\langle x',y'\rangle$ for $x \neq x'$. In other words, each top-level bucket only contains edges leaving a single node. Thus, answering rank queries for $\langle x,0\rangle$ only requires summing up the sizes of a number of top-level buckets, which is supported by the top level representation.

Although the universe size increases to $k'n$ from kn, since $k'n = kn(1+O(1/\sqrt{\lg n}))$, the increase in the space is only in the lower-order terms. With this additional power, we now support the remaining operations as follows:

- To find the degree of the node labeled x, we do the following. Find the rank of $\langle x+1,0\rangle$ and the rank of $\langle x,0\rangle$. The difference gives the degree of the node labeled x.

- To find the i-th child of the node labeled x, we do the following. Find the rank r of $\langle x,0\rangle$ and then do $select(r+i)$. The label returned by the *select* operation is the label of the i-th child of x.

- To find the local rank (i.e. rank with respect to the node) of the child labeled y of the node x, if exists, subtract the rank of $\langle x,0\rangle$ from $rank(\langle x,y\rangle)$, if $rank(\langle x,y\rangle) > 0$. □

6 Indexable Dictionary in Cell Probe Model

In this section we give of a representation of an indexable dictionary for a set S of size n from a universe of size m that uses $\mathcal{B}(n,m)+o(n)$ bits of space in cell probe model. In this model, time is measured as just the number of words (cells) accessed during an operation. All other computations are free.

Note that the $O(\lg\lg m)$ term exists in the proof of Theorem 3.3 because of the global range reduction (when $m > n^2$). In this section we show how this can be removed from the space complexity, in the cell probe model. We slightly modify the definition of small sets defined in the proof of Lemma 2.5 and call a set S small if $n \leq \sqrt{\lg M}$ (instead of $\sqrt{\lg N}$) and large otherwise.

We first prove the following lemma to represent a small set, in the cell probe model. This lemma is analogous to Lemma 2.4 for small sets without assuming access to the two functions.

LEMMA 6.1. *There is an indexable dictionary for a set $S \subseteq [m]$ of size $d \leq \sqrt{\lg m}$ using $d \lg m + d \lg d + O(d)$ bits in the cell probe model.*

Proof. We divide the $\lg m$ bit representation of each element into d^2 equal parts of size roughly $\frac{\lg m}{d^2}$. Since $d \leq \sqrt{\lg m}$ each part is non-empty. Then there exists a set of at most d of these parts such that every element of S is uniquely distinguished by these d parts. We store an implicit representation of this set of d parts using $\lg \binom{d^2}{d} = d \lg d + O(d)$ bits. For each element, we concatenate these d parts together to get a bitstring of length $\frac{\lg m}{d}$. If we list this bitstring for each of the d elements of S, we know that they are all distinct.

First we store these d parts for all the d elements of S (in sorted order) contiguously using $\lg m$ bits. Then we store the remaining parts of the elements consecutively in the sorted order of the elements. This gives a representation that takes $d \lg m + d \lg d + O(d)$ bits of space.

Given a query element, we can extract the d parts given by the representation of the set from the element (note that the time to extract is free in cell probe model). We then compare these d parts with the d parts of each element to find the unique match, if it exists. Since $d^2 \leq \lg m$, this can be done in constant time. If there is a match, we compare the remaining $d^2 - d$ parts of the element with those of the matched element (in the order in which they appear).

It is easy to see that the operations *rank* and *select* can be supported in constant time using this representation. □

Using the representation of Lemma 6.1 for small sets and of [18] for large sets, at the bottom level of the recursion in the proof of Theorem 3.3, the additive $O(\lg \lg m)$ terms can be removed from the bounds in Theorems 3.1 to 3.3. In particular, we get the following theorem.

THEOREM 6.1. *There is an indexable dictionary for a set $S \subseteq [m]$ of size n using $\mathcal{B}(n, m) + o(n)$ bits in the cell probe model.*

As an immediate corollary we get:

COROLLARY 6.1. *There is an indexable dictionary for a set $S \subseteq [m]$ of size n using at most $n \lceil \lg m \rceil$ bits in the cell probe model.*

7 Conclusions

We have given a static dictionary structure to store an n element subset of an m element universe, that takes the optimal (within a lower order term) $\mathcal{B}(n, m) + o(n) + O(\lg \lg m)$ bits of space and supports *membership*, *rank* (for those elements present) and *select* operations in constant time.

Using this structure, we have shown that a k-ary tree on n nodes can be represented using $\mathcal{C}(n, k) + o(n) + O(\lg \lg k)$ bits of space and supports all the navigational operations, except the subtree size of a given node, in constant time. Here $\mathcal{C}(n, k)$ is the information theoretically optimum number of bits required to represent a k-ary tree on n nodes. Applying our indexable dictionary, we also developed an optimal representation for a multiset of n elements from an m element universe using $\mathcal{B}(n, m + n) + o(n) + O(\lg \lg m)$ bits.

A variant of our dictionary representation gives a structure for the static dictionary problem in the cell probe model that takes the optimal (within a lower order term) $B(n, m) + o(n)$ in which membership (and select and rank operations) can be supported in constant time. This, in particular, means that n words (of size $\lg m$ bits) are sufficient to represent a static dictionary on n elements from an m element universe and answer membership queries.

References

[1] M. Ajtai, M. L. Fredman, and J. Komlós. Hash functions for priority queues. *Information and Computation*, 63:217–225, 1984.

[2] A.V.Aho, J.E.Hopcroft, J.D.Ullman. Data Structures and Algorithms. Addison-Wesley Publishing Company, Reading, Massachusetts, 1983.

[3] P. Beame and F. E. Fich. Optimal Bounds for the Predecessor Problem. *Proc. 31st ACM STOC*, pp. 295-304, 1999.

[4] D. Benoit, E. D. Demaine, J. I. Munro, and V. Raman. Representing trees of higher degree. In *Proceedings of WADS*, volume 1663 of *LNCS*, pages 169–180. Springer, 1999.

[5] D. Benoit, E. D. Demaine, J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Representing trees of higher degree. Manuscript, 2001.

[6] A. Brodnik and J. I. Munro. Membership in constant time and almost minimum space. *SIAM Journal on Computing*, 28(5):1628–1640, 1999.

[7] D. Clark. *Compact Pat Trees*. PhD thesis, University of Waterloo, 1996.

[8] D. Clark and J. I. Munro. Efficient suffix trees on secondary storage. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 383–391, 1996.

[9] J. J. Darragh, J.G. Cleary, I.H. Witten. Bonsai: A compact representation of trees. *Software – Practice and Experience*, 23(3):277–291, 1993

[10] P. Elias. Efficient storage retrieval by content and address of static files. *Journal of the ACM*, 21(2):246–260, April 1974.

[11] F. Fich and P. B. Miltersen. Tables should be sorted (on random access machines). In *Proceedings of WADS*, volume 955 of *LNCS*, pages 482–493. Springer, 1995.

[12] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM*, 31(3):538–544, July 1984.

[13] T. Hagerup. Sorting and searching on the word RAM. In *Proceedings of STACS*, volume 1373 of *LNCS*, pages 366–398. Springer, 1998.

[14] G. Jacobson. *Succinct Static Data Structures.* PhD thesis, Carnegie Mellon University, January 1989.

[15] J. I. Munro. Tables. In *Proceedings of FST & TCS*, volume 1180 of *LNCS*, pages 37–42. Springer, 1996.

[16] J. I. Munro, V. Raman, and S. S. Rao. Space efficient suffix trees. *Journal of Algorithms*, 39(2):205–222, 2001.

[17] R. Pagh. Low redundancy in dictionaries with o(1) worst case lookup time. In *Proceedings of ICALP*, volume 1644 of *LNCS*, pages 595–604. Springer, 1999.

[18] V. Raman and S. S. Rao. Static dictionaries supporting rank. In *Proceedings of ISAAC*, volume 1741 of *LNCS*, pages 18–26. Springer, December 1999.

[19] J. P. Schmidt and A. Siegel. The spatial complexity of oblivious k-probe hash functions. *SIAM Journal on Computing*, 19(5):775–786, 1990.

[20] R. E. Tarjan and A. C.-C. Yao. Storing a sparse table. *Communications of the ACM*, 22:606–611, 1979.

[21] A. C. Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, 1981.

Appendix

Proof of Lemma 2.5. Let $n_{i_j} = |S_{i_j}|$ and let $S^* = \cup_{j=1}^{s} S_{i_j}$. We first define a function f, which is a 'global' range reduction function.

If $M \leq N^2$ define f as $f(x) = x$. Otherwise, if $M > N^2$, then we find a hash function f given by $f(x) = (kx \bmod p) \bmod N^2$ for some prime $p \leq N^2 \lg M$ and $k \leq p$ which maps S^* 1-1 into the set $[N^2]$. The existence of such a function is guaranteed by FKS [12]. The function f, along with k^{-1} is represented using $O(\lg N + \lg\lg M)$ bits.

Note that choosing $h_{S_{i_j}} = f$ suffices if $n_{i_j} \geq \sqrt{\lg N}$, but if S_{i_j} is small, i.e., $n_{i_j} < \sqrt{\lg N}$ we need to reduce the range even further. We form $\lg N$ consecutive sets of the given collection of sets as a group[3]. Let S_i be the union of all elements in the small sets in the ith group. For the ith group, find a prime p_i such that the the function $g_i(x) = f(x) \bmod p_i$ is one-one on the set $f(S_i)$. Such a p_i whose value is at most $O(|S_i|^2 \lg ||f||)$ exists[12, 19]. Since $||f||$ is $O(N^2)$ and $|S_i| \leq (\lg N)^{3/2}$, we can represent p_i using $O(\lg\lg N)$ bits.

We store these primes, indexed by their group number in a separate table. Each prime is stored in a field of $b = \Theta(\lg\lg N)$ bits. If S_i is empty (i.e. there is no small set in the i-th group) then the table contains a string of b 0s in the entry corresponding to that group. The total space required by this table is $O(N \lg\lg N / \lg N)$ which is $o(N)$ bits. For a small set S_{i_j} the function $h_{S_{i_j}}$ which is required by Lemma 2.4 is precisely g_i if S_{i_j} belongs to the i-th group.

The functions q_S required in Lemma 2.4 are described below. To recover the original element x_l from $r_l = g_i(x_l)$, we need to store the following 'quotient' values $q_l = q_i(x_l)$ for $x_l \in S_i$:

$$q_l = ((x_l \text{ div } p)\lceil p/N^2 \rceil + (kx_l \bmod p) \text{ div } N^2)\lceil N^2/p_i \rceil + f(x_l) \text{ div } p_i.$$

From r_l and q_l and using k^{-1}, p and p_i one can obtain x_l in constant time. Observe that $||h_{S_{i_j}}|| + ||q_{S_{i_j}}|| \leq \lg M + 4$, which satisfies the hypothesis of Lemma 2.4.

[3] For easy access to the functions, we will in reality divide the range of indices into groups.