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

Rajeev Raman[†] Venkatesh Raman[‡] S. Srinivasa Rao[§]

Abstract

We consider the *indexable dictionary* problem which consists of storing a set $S \subseteq \{0, \ldots, m-1\}$ for some integer m, while supporting the operations of $\mathsf{rank}(x)$, which returns the number of elements in S that are less than x if $x \in S$, and -1 otherwise; and $\mathsf{select}(i)$ 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 present several extensions and applications of our dictionary structure including:

- an information-theoretically optimal representation of a k-ary cardinal tree that supports standard operations in constant time
- an optimal space multiset representation that supports (appropriate generalizations of) rank and select operations in constant time, and
- a representation of a sequence of n non-negative integers summing up to an integer m in $\mathcal{B}(n, m+n) + o(n)$ bits, to support partial sum queries in constant time.

1 Introduction

1.1 Motivation

Given a set S of n distinct keys from the universe $\{0, \ldots, m-1\}$, possibly the most fundamental data structuring problem that can be defined for S is the *dictionary* problem: to store S so that *membership* queries of the form "Is x in S?" can be answered quickly. In

*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`

[§]School of Computer Science, University of Waterloo, Waterloo N2L 3G1, Canada. `ssrao@uwaterloo.ca`

his influential paper [29], Yao considered the complexity of this problem and showed that the sorted array representation of S is the best possible for this problem, if one considers a suitably restricted class of representations. Since membership queries take $\Omega(\lg n)$ time to answer using a sorted array, a number of researchers have developed representations based on hashing that answer membership queries in constant time (see e.g. [29, 28, 14, 7, 24]).

However, one extremely useful feature present in the sorted array representation of S is that, given an index i, the i-th smallest element in S can be retrieved in constant time. Also, when the presence of an element x has been established, we also know its *rank*, i.e., the number of elements in S that are less than x. Schemes based on hashing work by "randomly scattering" keys, and do not intrinsically support such operations. It is natural to ask whether one can represent S in a way that combines the speed of hash tables with the additional functionality of sorted arrays. We therefore consider the problem of representing S to support the following operations in constant time:

$\mathsf{rank}(x, S)$: Given $x \in \{0, \ldots, m-1\}$, return -1 if $x \notin S$ and $|\{y \in S | y < x\}|$ otherwise, and

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

When there is no confusion, we will omit the set S from the description of these operations. We call this the *indexable dictionary* problem, and a representation for S where both these operations can be supported in constant time an *indexable dictionary representation.* A related problem was considered by Elias [12].

Our interest lies in *succinct* representations of S, whose space usage is close to the information-theoretic lower bound. Motivated by applications to very large data sets, as well as by applications to low-resource systems such as handheld and embedded computers, smart cards etc., there has been a renewal of interest in succinct representations of data [7, 6, 20, 17, 18, 22, 23, 24, 27]. In the context of this paper, the information-theoretic lower bound is obtained by noting that as there are $\binom{m}{n}$ subsets of size n from $\{0, \ldots, m-1\}$, one cannot represent an arbitrary set of n keys from $\{0, \ldots, m-1\}$ in fewer than $\mathcal{B}(n, m) = \left\lceil \lg \binom{m}{n} \right\rceil$ bits [1].

As $\mathcal{B}(n, m) \approx n(\lg m + \lg e - \lg n - \Theta(n/m)) + O(\lg n)$ (using Stirling's approximation), even a sorted array representation of S, which takes $n \lceil \lg m \rceil$ bits, can be significantly larger than the information-theoretic lower bound. Brodnik and Munro [7] were the first to give a succinct representation that supported constant-time membership queries. Pagh [24] improved the space bound to $\mathcal{B}(n, m) + o(n) + O(\lg \lg m)$ bits, while continuing to support membership queries in constant time. Raman and Rao [26] considered *dictionaries with rank*, which support constant-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 also 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. All the above papers [7, 24, 26] assume the standard *word RAM* model with word size $\Theta(\lg m)$ bits [1, 19]; unless specified otherwise this is our default model.

[1] $\lg x$ denotes $\log_2 x$.

1.2 Our results

In this paper, we give an indexable dictionary representation that requires $\mathcal{B}(n,m)+o(n)+O(\lg\lg m)$ bits to store a set of size n from $\{0,\ldots,m-1\}$. In the *cell probe* model [29] with word size $\Theta(\lg m)$, a variant of this data structure requires $\mathcal{B}(n,m)+o(n)$ bits. The significance of this modest improvement in space usage is as follows. Since $\mathcal{B}(n,m)+o(n)\leq n\lceil\lg m\rceil$ for all n larger than a sufficiently large constant, this result shows that n words of $\lceil\lg m\rceil$ bits suffice to answer membership queries in constant time on a set of size n. This answers a question raised by Fich and Miltersen [13] and Pagh [24]. By contrast, Yao showed that if the n words must contain a permutation of S, then membership queries cannot be answered in constant time [29]. We now give two applications of indexable dictionaries.

- 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 $\mathcal{C}(n,k)=\left\lceil\lg\left(\frac{1}{kn+1}\binom{kn+1}{n}\right)\right\rceil$ [16]. Note that $\mathcal{C}(n,k)\approx n(\lg(k-1)+k\lg\frac{k}{k-1})$ which is close to $n(\lg k+\lg e)$, as k grows. Benoit et al. [6] gave a cardinal tree structure that takes $(\lceil\lg k\rceil+2)n+o(n)+O(\lg\lg k)=\mathcal{C}(n,k)+\Omega(n)$ bits and answers queries asking for parent, i-th child, child with label i, degree and subtree size in constant time.

  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. All the above results on cardinal trees use the word RAM model with a word size of $\Theta(\lg(kn))$ bits.

- Let M be a multiset of n numbers from $\{0,\ldots,m-1\}$. We consider the problem of representing M to support the operations rankm and $\mathsf{selectm}$ that are natural generalisations of rank and select to multisets. 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 $\{0,\ldots,m+n-1\}$ [12]. However, if we transform a multiset into a set by this mapping, then rankm and $\mathsf{selectm}$ do not appear to translate into rank and select operations on the transformed set. Using some additional ideas, we obtain a multiset representation that takes $\mathcal{B}(n,m+n)+o(n)+O(\lg\lg m)$ bits, and supports rankm and $\mathsf{selectm}$ in constant time. This result assumes a word size of $\Theta(\lg(m+n))$ bits.

  Elias [12] previously considered the problem of succinctly representing multisets while supporting membership and $\mathsf{selectm}$. He considered the *bit-probe* model rather than the word RAM model, and was concerned with average-case behaviour for a randomly chosen multiset. Thus, his results are incomparable to ours.

We also give a subroutine that appears to be of independent interest. Given a bit-vector σ of m bits, define the following operations, for $b\in\{0,1\}$: (a) $\mathsf{rank}_b(i)$ – count the number of b's before the position i in σ, and (b) $\mathsf{select}_b(i)$ – find the position of the i-th b in σ. One can also view σ as the characteristic vector of a subset S of n keys from $U=\{0,\ldots,m-1\}$, and define a *fully indexable dictionary (FID)* representation of S to be one that supports the operations $\mathsf{rank}(x,S)$, $\mathsf{select}(i,S)$, $\mathsf{rank}(x,\bar{S})$ and $\mathsf{select}(i,\bar{S})$ all in constant time, where

$\bar{S} = U \setminus S$ is the complement of the set S. It is easy to see that an FID representation is functionally equivalent to a bit-vector supporting rank_b and select_b, for $b \in \{0, 1\}$, queries in constant time; for example, $\mathsf{rank}_1(j)$ is given by $\mathsf{rank}(j, S)$ if the j-th bit is a 1, and by $j - \mathsf{rank}(j, \bar{S}) - 1$ otherwise. It is shown in [8, 23] how to represent σ in $m + o(m)$ bits and support these four queries in constant time. This data structure is a fundamental building block in a large number of succinct data structures [28, 24, 17, 23, 5, 21].

Extending a result due to Pagh [24], we give an FID representation for S that takes $\mathcal{B}(n, m) + O((m \lg\lg m)/\lg m)$ bits. This is always at most $m + o(m)$ bits, but it may be substantially less: for example, whenever $m/\sqrt{\lg m} \le n \le m(1 - 1/\sqrt{\lg m})$, the space usage is at most $\mathcal{B}(n, m) + o(n) = (1 + o(1))\mathcal{B}(n, m)$ bits. We give the following application of this result:

- We can store a multiset M of n values from $\{0, \ldots, m-1\}$ to support $\mathsf{selectm}$ (but not rankm) in constant time using $\mathcal{B}(n, m+n) + o(n)$ bits. Another way of stating this result is that we can represent a sequence of n non-negative integers $X = x_1, \ldots, x_n$, such that $\sum_{j=1}^{n} x_j = m$, so that the query $\mathsf{sum}(i, X)$, which returns $\sum_{j=1}^{i} x_j$, can be answered in constant time using $\mathcal{B}(n, m + n) + o(n)$ bits.

The problem of representing integers compactly so that their prefix sums can be recovered has been studied by a number of researchers including [12, 17, 18, 24, 28]. Our solution is more space-efficient than all of these. The result of Grossi and Vitter [17, Lemma 2], which is based on Elias's ideas, is the previously most space-efficient one and requires $n(\lceil \lg m \rceil - \lfloor \lg n \rfloor + 2) + o(n)$ bits to represent n non-negative integers adding up to m, where $m \ge n$. In most cases, this will be $\Theta(n)$ bits more than optimal. When n and m are not powers of 2, the ceilings and floors are a source of non-optimality; for example, take $m = n$ with m not a power of 2; Grossi and Vitter's method requires $3n + o(n)$ bits in the worst case, as opposed to the lower bound of $\mathcal{B}(n, 2n) = 2n - O(\lg n)$ bits. Another source of non-optimality is that the constant 2 is not optimal; for example, take $m = cn$ where n and m are powers of 2 and $c > 1$. Grossi and Vitter's method requires $(2 + \lg c)n + o(n)$ bits in the worst case, which can be easily shown to be at least $(2 - (1 + c)\lg((1 + c)/c))n = \Omega(n)$ bits more than optimal (the difference tends to $(2 - \lg e)n$ as c increases). On the other hand, our representation is always within $o(n)$ bits of optimal.

It is important to note that, appearances notwithstanding, some of the space bounds above may actually be very far from optimal. For example, consider the space bound of $\mathcal{B}(n, m) + o(n) + O(\lg\lg m)$ bits for storing a set S of size n from $\{0, \ldots, m-1\}$ in an indexable dictionary representation. If $n \le m/2$, $\mathcal{B} = \mathcal{B}(n, m) \ge \max\{n, \lg m\}$ and this space bound is indeed $\mathcal{B}$ plus lower-order terms. However, as n gets very close to m, $\mathcal{B}$ can be much smaller than the $o(n)$ term. If we only want to answer membership queries, we can assume $n \le m/2$ without loss of generality: if S has more than $m/2$ elements then we store its complement and invert the answers. However, in the indexable dictionary problem, it is not clear how answering rank and select queries on a set could help us to answer these queries on its complement in constant time. In fact, we note that if we could store a set S in $\mathcal{B}^{O(1)}$ bits, for all n and m, and support select (or rank) in constant time, then we could support $\mathsf{fullrank}$ queries on S also using $\mathcal{B}^{O(1)}$ bits of space. Here $\mathsf{fullrank}(x, S)$ returns the rank of x in S for any $x \in U$. It is known that in general, $\mathsf{fullrank}$ queries cannot be answered in constant time on the RAM model (or even on the cell probe model) while using $n^{O(1)}$ words of $(\lg m)^{O(1)}$ bits each [4]. Thus, many of our space bounds are of necessity non-optimal in some cases; one exception is the space bound for k-ary trees, which is optimal for all $k \geq 2$.

1.3 Techniques used

The main ingredient in our indexable dictionary representation is *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 [7] 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 prefix sums of bucket sizes that not only provides the extra functionality needed for supporting rank and select, but also uses significantly less space. The related technique of *quotienting* [24, 10] stores only the "quotients" of keys that are mapped to a bucket by a standard hash function (e.g. those of [14]). 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.

Other ideas relevant to the indexable dictionary representation are range reduction ([14] and others), distinguishing bits ([2] and others) and techniques for compactly representing hash functions for several subsets of a common universe developed in [6]. Our k-ary tree representation uses a numbering of the tree nodes that allows us to dispense with an explicit encoding of the tree structure, a feature shared with the representation of [11].

1.4 Organization of the paper

The remainder of this paper is organized as follows. In Section 2, we give some building blocks that will later be used in our main results. Extending the dictionary with rank of [26, 6], we first give a simple indexable dictionary that uses about $2n \lg n$ bits more than necessary. Then we show the connection between fully indexable dictionaries and prefix sum structures and give some simple representations for both. These are then used in Section 3, coupled with MSB bucketing, to obtain an improved result on indexable dictionaries, which reduces space wastage to about $O(n)$ bits.

In Section 4, we first develop a $\mathcal{B}(n,m) + O((m \lg\lg m)/\lg m)$-bit fully indexable dictionary representation for dense sets (when m is $O(n\sqrt{\lg n})$), extending a result of Pagh [24]. Using this and our result in the earlier section, we obtain our main result: an indexable dictionary taking $\mathcal{B}(n,m) + o(n) + O(\lg\lg m)$ bits. In Section 5, we remove the $O(\lg\lg m)$ term in the space bound by moving to the cell probe model, giving a representation that takes $\mathcal{B}(n,m) + o(n)$ bits. Section 6 gives some applications of our succinct dictionaries to representations of multiple dictionaries, k-ary trees, multisets and prefix sums. Section 7 makes some observations about the difficulty of achieving optimal space for all values of the input parameters. Section 8 recapitulates the main results and gives some open problems.

2 Preliminaries

In this section, we first establish connections between FIDs and prefix sums, and we end with simple representations of multiple indexable dictionaries and prefix sums.

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 $\{0, 1, \ldots, m-1\}$.

2.1 Fully Indexable Dictionaries and Searchable Prefix Sums

Given a set $S \subseteq U$, recall that 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. FIDs are essential to our data structure as they are intimately related to operations on prefix sums, as we note below.

2.1.1 Searchable Prefix Sums

Given a sequence X of n non-negative integers $x_1, \ldots, x_n$ such that $\sum_{i=1}^{n} x_i = m$, the *searchable prefix sum problem* is to find a representation of this sequence to support the following operations in constant time:

$\mathsf{sum}(i, X)$**:** Given $i \in \{1, \ldots, n\}$, return $\sum_{j=1}^{i} x_j$.

$\mathsf{pred}(x, X)$**:** Given $x \in [m]$, return $\max\{i \leq n | \sum_{j=1}^{i} x_j < x\}$.

We now make the connection between FIDs and the searchable prefix sums problem.

Lemma 2.1 *Suppose there is an FID representation for any given set $S \subseteq U$ using $f(|S|, |U|)$ bits. Then given a sequence X of non-negative integers $x_1, x_2, \ldots, x_n$ such that $\sum_{i=1}^{n} x_i = m$ there is a structure to represent X using $f(n, m+n)$ bits that supports* sum *and* pred *operations in constant time.*

Proof: Consider the following $m+n$ bit representation of the sequence X [12]. For $i = 1$ to n, represent x_i by x_i 0s followed by a 1. Clearly this representation takes $m+n$ bits since it has m 0s and n 1s. View this bit sequence as the characteristic vector of a set S of n elements from the universe $[m+n]$. Represent S as an FID using $f(n, m+n)$ bits. It is easy to verify that $\mathsf{pred}(x, X) = \mathsf{select}(x, \bar{S}) - x + 1$ and $\mathsf{sum}(i, X) = \mathsf{select}(i, S) - i + 1$. (Recall that $[m+n]$ begins with 0.) □

Lemma 2.2 ([20, 21] (see also [23])) *Given a set $S \subseteq [m]$ there is an FID on S that requires $m + o(m)$ bits.*

Proof: Consider the characteristic vector of S, which is a bit-vector of length m. It is shown in [8, 23] how to represent this bit-vector using $m+o(m)$ bits, to support the queries $\mathsf{rank}_b(i)$ and $\mathsf{select}_b(i)$ in constant time. It is easy to verify that these operations on the characteristic vector suffice to support FID operations on S. □

The following lemma follows from Lemma 2.1 and Lemma 2.2.

Lemma 2.3 *A sequence S of n non-negative numbers whose total sum is m can be represented using $m+n+o(m+n)$ bits to support* sum *and* pred *operations in constant time.*

2.2 A Simple Indexable Dictionary

We now give a simple indexable dictionary representation for a set $S \subseteq [m^*]$, based upon perfect hashing schemes for membership [14, 27, 24]. The perfect hashing schemes above begin with finding a *universe reduction* function $f : [m^*] \to [|S|^2]$ such that f is $1-1$ on S. A problem with such an approach is that f requires $\Omega(\lg \lg m^*)$ bits to represent, which can be a significant overhead for sets which are very small, but nonetheless not constant-sized. This is particularly bad if we need to store several sets in the data structure, as we pay the overhead repeatedly for each set. To reduce this overhead we use the idea of [6], which is to note that we do not need f to bring the universe size as far down as $|S|^2$ for small sets, thereby allowing the same f to be used for several (small) sets.

Thus, it makes sense to talk about representing the set S, but excluding the space cost of representing a universe-reduction function. In the following lemma, which is a simplification and extension of one from [6], we use this approach. Here h_S is the universe-reduction function and q_S is a "quotient" function, which gives the information thrown away during the universe reduction and is used to recover x given $h_S(x)$.

Lemma 2.4 *Let $m^*, n^* \geq 1$ be two given integers, and let $S \subseteq [m^*]$ be a set of size at most n^*. Suppose we have access to two functions h_S and q_S, 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 $|S| > \sqrt{\lg n^*}$ and $O((\lg n^*)^c)$ for some constant $c > 0$ otherwise.*
4. *$\lceil \lg ||h_S|| \rceil + \lceil \lg ||q_S|| \rceil = \lg m^* + O(1)$.*

Then we can represent S using $|S|(\lg m^* + \lg |S| + O(1))$ bits and support rank *and* select *in $O(1)$ time. This assumes a word size of at least $\lg \max\{m^*, n^*\}$ bits, and access to a pre-computed table of $o(n^*)$ bits and a constant of $O(\lg n^*)$ bits that depends only on $||h_S||$, and that m^* and n^* are known to the data structure.*

Proof: Let $l = |S|$ and suppose that S contains the elements $x_1 < x_2 < \ldots < x_l$.

If $l \leq \sqrt{\lg n^*}$ then we write down $h_S(x_1), \ldots, h_S(x_l)$ in fields of $b = \lceil \lg ||h_S|| \rceil$ bits each, followed by $q_S(x_1), \ldots, q_S(x_l)$ in fields of $\lceil \lg ||q_S|| \rceil$ bits each. This requires $|S|(\lg m^* + O(1))$ bits. To compute rank(x) we calculate $h_S(x)$ and look for a match in $h_S(x_1), \ldots, h_S(x_l)$. This can be done in $O(1)$ time using standard techniques [25, 15, 3], provided we have available the integer constant k that contains 1s in bit positions $0, b, 2b, \ldots, b \cdot \lfloor \lg n^*/b \rfloor$, as well as tables that enable us to compute, for every integer x of $\lg n^*$ or fewer bits, the index of the most significant bit that is set to 1 (or, equivalently to compute $\lfloor \lg x \rfloor$). If we are unable to find an index i such that $h_S(x) = h_S(x_i)$, we return -1, otherwise we verify whether $q_S(x) = q_S(x_i)$. If so, return $i-1$, otherwise return -1. To compute $\mathsf{select}(i)$, reconstruct x_i from the values $h_S(x_i)$ and $q_S(x_i)$ and return it.

If $l > \sqrt{\lg n^*}$, then let $S' = \{h_S(x) | x \in S\}$. We create a *minimal* perfect hash function $f : [||h_S||] \to [l]$ that is $1-1$ on S'. As shown in [27, 18], there exists such a function f that can be evaluated in $O(1)$ time and that can be represented in $O(l + \lg\lg ||h_S||) = O(l + \lg\lg n^*) = O(l)$ bits. We also store two tables of size l. In the first table R, for $1 \le i \le l$, we store the value i in the location $f(h_S(x_i))$ using a total of $l \lceil \lg l \rceil$ bits. In the second table X, we store x_i in location $i-1$, for $i = 1, \ldots, l$. Now to answer $\mathsf{rank}(x)$, we calculate $j = R[f(h_S(x))]$ and check if $x = x_j$: if so, then $\mathsf{rank}(x)$ is $j-1$ and $\mathsf{rank}(x)$ is -1 otherwise. Supporting select is trivial since we have stored the x_is in sorted order in X. $\square$

The following lemma from [6] gives the space savings obtained by combining universe reduction functions for different sets:

Lemma 2.5 *[6] Let n^*, m^* be as in Lemma 2.4, and let $0 \le i_1 < i_2 < \ldots < i_s < n^*$ be a sequence of integers. Let $S_{i_1}, S_{i_2}, \ldots, S_{i_s}$ be subsets of $[m^*]$ such that $\sum_{j=1}^{s} |S_{i_j}| \le n^*$. Then 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, which can be represented in $o(n^*) + O(\lg\lg m^*)$ bits in such a way that given i_j we can access $h_{S_{i_j}}$ and $q_{S_{i_j}}$ in constant time.*

3 Saving $n \lg n$ bits using MSB Bucketing

In this section, we first give a representation that takes about $n\lceil \lg m \rceil$ bits to represent a set of size n and to support rank and select operations in $O(1)$ time (Theorem 3.1). We then use this representation to store multiple independent (but not necessarily disjoint) dictionaries efficiently (Lemma 3.1).

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: Our construction algorithm partitions 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.

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 $n \le d$, then we simply write down the elements of S using $n \lceil \lg m \rceil$ bits, and we are done.

Otherwise, if $n > d$, we partition the elements of S according to their top $\lceil \lg n \rceil$ bits. This partitions S into $y = 2^{\lceil \lg n \rceil} \le 2n$ sets denoted by $S_0, \ldots, S_{y-1}$, where S_i consists of the last t bits of all keys in S whose most significant $\lceil \lg n \rceil$ bits have value i, for $i \in [y]$. We store a representation of the sizes of these y sets which takes $n + y + o(n) \le 3n + o(n)$

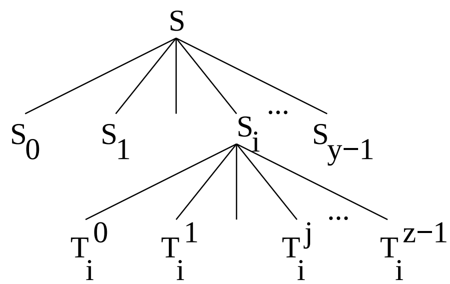

Figure 1: Two level MSB bucketing

bits using Lemma 2.3 and pad this out to $4n$ bits. The representation of S is obtained by concatenating these $4n$ bits with the representations of each of the S_i's, for $i \in [y]$.

The representation of S_i, for $i \in [y]$, is obtained as follows. Let $n_i = |S_i|$. If $n_i \leq d$, we write down the elements of S_i using $n_i t$ bits, and pad the output out to $n_i(t+4+c)$ bits. Otherwise, we again partition the elements of S_i into $z = 2^{\lceil \lg n_i \rceil}$ sets according to their top $\lceil \lg n_i \rceil$ bits, denoted as $T_i^0, \ldots, T_i^{z-1}$. We store a representation of the sizes of these z sets and pad this out to $4n_i$ bits. Again, the representation of S_i is the concatenation of these $4n_i$ bits with the representations of each of the T_i^js, for $j \in [z]$.

The representation of T_i^j, for $j \in [z]$, is obtained as follows. If $|T_i^j| \leq d$, then we write down its elements using $|T_i^j|(t - \lceil \lg n_i \rceil)$ bits, and pad the output out to $|T_i^j|(t+c)$ bits. Otherwise, we store it using the representation of Lemma 2.4 (with $m^* = 2^{t - \lceil \lg n_i \rceil}$ and $n^* = n$), padding this output out to $|T_i^j|(t+c)$ bits if necessary. (Note that the representation of T_i^j using Lemma 2.4 takes $|T_i^j|(t - \lceil \lg n_i \rceil + \lg |T_i^j| + O(1))$ bits. Thus it is enough to choose c to be equal to the constant in the $O(1)$ term to guarantee that this is at most $|T_i^j|(t+c)$ bits.)

When S_i is partitioned, its representation takes $4n_i + \sum_{j=0}^{z-1} |T_i^j|(t+c) = 4n_i + n_i(t+c) = n_i(t+4+c)$ bits. Thus the representation of S_i takes $n_i(t+4+c)$ in either case. Hence the length of the representation of S, when it is partitioned, is $4n + \sum_{i=0}^{y-1} n_i(t+4+c) = 4n + n(t+4+c) = n(t+8+c)$ bits. Thus, in either case, S takes $n(t+8+c)$ bits. This is at most $n \lceil \lg m \rceil$ bits, for sufficiently large d.

We now describe how the computation of rank proceeds; select works in a similar way. If $n \leq d$, we apply the trivial algorithm and return. Otherwise, we consider the first $4n$ bits of the representation S, which contains the representation of the sequence σ of the sizes of the buckets S_i, $i \in [y]$. We extract the the top $\lceil \lg n \rceil$ bits of the current key[2]; suppose that these bits have value i. Using Lemma 2.3, we calculate $\rho = \mathsf{sum}(i-1, \sigma)$ and $\rho' = \mathsf{sum}(i, \sigma)$ in $O(1)$ time; note that $\rho' - \rho$ is the size of the set S_i to which the current key belongs. The start of the representation of S_i is also easy to compute: it starts $4n + \rho(t+4+c)$ bits from the start of the representation of S. We then remove the top $\lceil \lg |T| \rceil$ bits from the query key, add the rank of the resulting key in the set S_i to ρ and return. Thus the problem reduces to finding the rank of a key in some set S_i.

If $|S_i| \leq d$, then we apply the trivial algorithm to find the rank of a key in S_i. Otherwise, we apply the similar algorithm as above to reduce the problem to finding the rank of a key

[2]Standard techniques allow us to calculate $\lceil \lg x \rceil$ in constant time [15].

in some set T_i^j. Again, if $|T_i^j| \le d$, then we apply the trivial algorithm to find the rank. Otherwise, since T_i^j is stored using the representation of Lemma 2.4, we can support rank in constant time. The overall computation is clearly constant-time.

It is easily verified that $n^* = n$ is an appropriate choice for all applications of Lemma 2.4 above. We now verify that the additional space required (in terms of the pre-computed table and constants) is not excessive. Firstly, the pre-computed table is of size $o(n^*) = o(n)$ bits and is common to all applications of Lemma 2.4. At most $O(\lg n)$ constants are required, one for each possible value of $b = \lceil \lg ||h_S|| \rceil$, which require negligible space.

We now discuss the use of Lemma 2.5 to represent the functions for all the base-case sets. The lemma requires 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. Finally, the space required for representing the functions is $o(n) + O(\lg\lg m)$ bits. This completes the proof of the Theorem 3.1. □

The following lemma is an easy extension of Theorem 3.1.

Lemma 3.1 *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 operations* rank(x, S_i) *and* select(j, S_i) *can be supported in constant time for any $x \in [m], 1 \le j \le n$ and $1 \le i \le s$. This requires that we have access to a constant-time oracle which returns the prefix sums of the n_i values.*

Proof: If we apply Theorem 3.1 directly to each set S_i we get a representation taking $\sum_{i=1}^{s} (n_i \lceil \lg m \rceil + o(n_i) + O(\lg\lg m)) = n\lceil \lg m \rceil + o(n) + O(s \lg\lg m)$ bits, that supports rank and select on each set in $O(1)$ time. The beginning of the representation of each set can be calculated using the oracle supporting the prefix sum queries in constant time. To get the claimed bound, we apply the algorithm of Theorem 3.1 to represent each S_i, but we modify it so that Lemma 2.5 is used only once for all sets S_i. The only change this causes is that we need a global numbering (using indices bounded by n) of all base-case sets created when applying the algorithm of Theorem 3.1 to the S_i's. Recall that when applying the algorithm of Theorem 3.1 to a particular set S_i, we give each base-case set that is created a 'local' number bounded by n_i. Thus, an appropriate global number for a base-level set created when applying the algorithm of Theorem 3.1 to S_i is just its local number plus $\sum_{j=1}^{i-1} n_j$. This gives the claimed bound. □

4 Obtaining Sublinear Lower Order Term

In this section, we develop the main result of the paper, namely, a representation for an indexable dictionary taking $\mathcal{B}(n, m) + o(n) + O(\lg\lg m)$ bits of space. We begin by observing that the bound of Theorem 3.1 is better than claimed: it is actually $\mathcal{B}(n, m) + O(n + \lg\lg m)$ bits. The constant factor in the $O(n)$ term can be improved by means of one more level of MSB bucketing, as follows. We place the keys into $2^{\lfloor \lg n \rfloor}$ buckets based upon the first $\lfloor \lg n \rfloor$ bits of each element. We represent the sizes of these buckets using at most $2n + o(n)$ bits via Lemma 2.3. This partitions the given set into multiple (up to n) sets which contain keys of $\lceil \lg m \rceil - \lfloor \lg n \rfloor$ bits each; the collection of sets is then represented using the structure of Lemma 3.1. The resulting dictionary takes at most $n(\lceil \lg m \rceil - \lfloor \lg n \rfloor + 2) + o(n) + O(\lg \lg m)$ bits and supports rank and select in constant time.

Recalling the discussion on representing prefix sums in the introduction, this bound is also non-optimal by $\Theta(n)$ bits in many cases. In addition to redundancy caused when m and n are not powers of 2, the constant 2 is not optimal. For example, when $m = cn$ for some constant $c > 2$, the disparity in this case is $(2 - c\lg(c/(c-1)))n$ bits, which tends again to about $(2 - \lg e)n$ bits for large c. To bring the linear term of the space bound closer to optimal, we place the keys into $\Theta(n\sqrt{\lg n})$ buckets; this will also enable us to 'remove' the ceilings and floors in the bound. However, using a super-linear number of buckets uses too much space if we use Lemma 2.3 to represent their sizes. Hence, we now develop a much more space-efficient alternative to Lemma 2.3, by giving more space-efficient FIDs. In particular, we show the following lemma which is an extension of [24, Proposition 4.3].

4.1 Fully Indexable Dictionaries for Dense Sets

Lemma 4.1 *Given a set $S \subseteq [m]$, $|S| = n$, there is an FID on S that requires $\mathcal{B}(n,m) + O(m \lg \lg m / \lg m)$ bits of space.*

Proof: We divide the universe $[m]$ into $p = \lceil 2m/\lg m \rceil$ *blocks* of $u = \left\lfloor \frac{1}{2} \lg m \right\rfloor$ numbers each, with the i-th block $U_i = \{(i-1)u, \ldots, iu-1\}$, for $1 \le i \le p-1$, and $U_p = \{(p-1)u, \ldots, m\}$. Let $S_i = S \cap U_i$ and $n_i = |S_i|$. The set S_i is represented implicitly 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. S is represented by concatenating the representations of the S_i's; the length of this representation of S is at most $\mathcal{B}(n,m) + O(m/\lg m)$ bits, as shown in [7].

To enable fast access to the representations of the S_is, we store two arrays of size p. 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 $\mathcal{B}(n_i, u)$; since $\mathcal{B}(n_i, u) \le u$ 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 [24, Proposition 4.2] or [28], in $O(m \lg \lg m / \lg m)$ bits, such that the i-th prefix sum is calculated in $O(1)$ time. We also store precomputed tables to support rank and select queries on an arbitrary set S_i given its size and its implicit representation. These tables require $O(m^{1-\epsilon})$ bits of space for some fixed positive constant $\epsilon < 1$.

To find $\mathsf{rank}(x)$ we proceed as in [24]: 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.

To support select we do the following. We store an array C of size $q = \lceil n/(\lg p)^2 \rceil$ such that for $j = 1, \ldots, q$, $C[j]$ stores the index $l \le p$ such that $\Sigma_{i=1}^{l-1} n_i < j \lfloor (\lg p)^2 \rfloor \le \Sigma_{i=1}^{l} n_i$. The array C takes $O(n/\lg p) = O(n/\lg m)$ bits and allows $\mathsf{select}(j \lfloor (\lg p)^2 \rfloor)$ for $j = 1, \ldots, q$ to be answered in $O(1)$ time, as follows. Letting $k = C[j]$, we use the partial sums of B to extract the representation of S_k, use the partial sums of A to calculate $s = |S_1 \cup \ldots \cup S_{k-1}|$, and use table lookup to return the $(j \lfloor (\lg p)^2 \rfloor - s)$-th element from S_k as the final answer.

To support select for arbitrary positions, we follow the ideas of [8, 23]. Letting $C[0] = 0$, for $i = 1, \ldots, q$, we define the i-th *segment* as $\cup_{j=C[i-1]+1}^{C[i]} U_j$; i.e., the part of the universe that lies between two successive indices from C. As $C[i] > C[i-1]$ for all i, the segments are disjoint. We call a segment *dense* if its size is at most $(\lg p)^4$ and *sparse* otherwise. For each sparse segment, we explicitly list (in sorted order) the elements of S that lie in that segment. The space required to represent the elements of S that lie in a sparse segment is therefore $(\lg p)^2 \cdot \lg m$, but since there are at most $m/(\lg p)^4$ sparse segments, this adds up to $O(m/\lg m)$ bits overall.

For a dense segment, we construct a complete tree with branching factor $\lceil \sqrt{\lg p} \rceil$, whose leaves are the blocks that constitute this segment. Since the number of leaves is $O((\lg p)^3)$, the depth of this tree is constant. At each node of this tree, we store an array containing the number of elements of S in each of its child subtrees. If the tree for a dense segment has k leaves, the space usage for this tree is $O(k \lg\lg p)$ bits. As segments are disjoint and the total number of blocks is $O(m/\lg m)$, this adds up to $O(m \lg\lg m/\lg m)$ bits overall. We store explicit pointers to the beginning of the representation of each segment, which takes $O(m/\lg m)$ bits as there are only $O(m/(\lg m)^2)$ segments.

To compute $\mathsf{select}(i)$ we first identify the segment in which the i-th element can be found. Letting $k_1 = C[\lceil i/(\lg p)^2 \rceil]$, by inspecting the prefix sums of A at positions k_1 and $k_1 + 1$ one can determine whether the i-th element belongs to the segment beginning at k_1 or $k_1 + 1$. Suppose it belongs to the segment σ. Using the prefix sums of A, we determine the rank of the element to be selected in σ. If σ is sparse we read the required element directly from a sorted array. Otherwise, if σ is dense, we start at the root of the tree corresponding to σ 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 via table lookup using tables of negligible size, as the array at each node takes $O(\sqrt{\lg p} \lg\lg p) = o(\lg m)$ bits. Thus, in constant time we reach a leaf that corresponds to some block S_j which is known to contain the element sought. We find the number of elements s 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 prefix sum structure for the array B and find the position l of the $(i-s)$-th element in the representation of S_i using a table lookup.

Now we consider supporting rank and select operations on $\bar{S}$. Again letting $\bar{S}_i = \bar{S} \cap U_i$ and $\bar{n}_i = |\bar{S}_i|$, we observe that $\bar{n}_i = u - n_i$, and so the prefix sums of A suffice to answer prefix sum queries on the $\bar{n}_i$s. Likewise, the implicit representation of S_i is also an implicit representation of $\bar{S}_i$ and the concatenated representations of the S_is is also an implicit representation of $\bar{S}$ that takes only $\mathcal{B}(n, m) + O(m/\lg m)$ bits, from which the representation of a single $\bar{S}_i$ can be retrieved in $O(1)$ time using the array B. Thus, answering rank queries on $\bar{S}$ requires no additional information except new tables (of negligible size) for performing rank and select on the implicit representations of the $\bar{S}_i$s.

To answer select queries on $\bar{S}$, we create an array $\bar{C}$ which is analogous to the array C, and which partitions the universe anew into segments. Selecting elements from $\bar{S}$ in these segments is done as before, with trees for dense segments and sorted arrays for sparse segments. This requires $O(m \lg\lg m/\lg m)$ additional auxiliary space. $\square$

Remark: By replacing the implicit representations of the S_i's with the characteristic vector of set S, we get a representation of a bit-vector of length m that takes $m+O(m\lg\lg m/\lg m)$ bits and supports rank_b and select_b queries, for $b \in \{0,1\}$ (defined in Section 1.2), in constant time. This improves the lower-order term in space of the earlier known structures [8, 23] from $O(m/\lg\lg m)$ to $O(m\lg\lg m/\lg m)$.

It immediately follows that:

Corollary 4.1 *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 4.1 and Lemma 2.1. Note that $\mathcal{B}(n, m+n)$ is the information theoretic minimum number of bits to represent a multiset of n elements from $[m]$.

Corollary 4.2 *If m is $O(n\sqrt{\lg n})$, then a sequence S of n non-negative numbers whose total sum is m can be represented using $\mathcal{B}(n, m+n)+o(n)$ bits to support* sum *and* pred *operations in constant time.*

4.2 Optimal Bucketing for Sparse Sets

Now we use Corollary 4.2 to prove our main result:

Theorem 4.1 *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.*

Proof: First, if $m < 4n\sqrt{\lg n}$ then we use Corollary 4.1, which establishes the result. If $m \geq 4n\sqrt{\lg n}$, we choose an integer $l > 0$ such that $n\sqrt{\lg n} \leq \lfloor m/2^l \rfloor < 2n\sqrt{\lg n}$. We now group the keys based upon the mapping $g(x) = \lfloor x/2^l \rfloor$. Let $r = \lfloor (m-1)/2^l \rfloor$. We "partition" S into sets B_i, for $i = 0, \ldots, r$ where $B_i = \{x \bmod 2^l \mid x \in S \text{ and } g(x) = i\}$. Let $b_i = |B_i|$, for $i = 0, \ldots, r$. We represent the sequence $B_{top} = (b_0, \ldots, b_r)$ using the structure of Corollary 4.2 taking $\mathcal{B}(n, r+n+1) + o(n)$ bits, which supports sum and pred on B_{top} in constant time. As $r = \Theta(n\sqrt{\lg n})$, the space usage is $\mathcal{B}(n,r) + o(n)$ bits.

The overall representation is the following. First we represent B_{top} as above. Then we represent each of the B_i's using the structure of Lemma 3.1. The total space used will be $nl + \mathcal{B}(n,r) + o(n) + O(\lg\lg m)$ bits. Note that $\mathcal{B}(n,r) = n\lg(er/n) + o(n)$ as $r = \Theta(n\sqrt{\lg n})$, and so $nl + \mathcal{B}(n,r) = nl + n\lg(me/(2^l n)) + o(n) = \mathcal{B}(n,m) + o(n)$. Thus, the overall space bound is as claimed. The computations of rank and select proceed essentially as in Theorem 3.1, except that we use Corollary 4.2 instead of Lemma 2.3. $\square$

5 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. We first prove a lemma analogous to Lemma 2.4 without assuming the access to the functions h_S and q_S.

Lemma 5.1 *There is an indexable dictionary for a set $S \subseteq [m]$ of size n that uses $n(\lg m + \lg n + O(1))$ bits in the cell probe model.*

Proof: Let $x_1 < x_2 < \ldots < x_n$ be the elements of S.

If $n \geq \sqrt{\lg m}$, then we first store the given set S in an array A in increasing order, which takes $n \lg m + O(n)$ bits of space. As in Lemma 2.4, we find a minimal perfect hash function f for S and store it using $O(n + \lg\lg m) = O(n)$ bits (since $n \geq \sqrt{\lg m}$). We then store a table T with $T[f(x_i)] = i$. This requires $n \lg n + O(n)$ bits. To answer $\mathsf{rank}(x)$, we calculate $j = T[f(x)]$ and check if $x = x_j$; if so we return $j - 1$, otherwise return -1. Supporting select is straightforward, as we store the elements in sorted order in A.

Otherwise, if $n < \sqrt{\lg m}$, we divide the $\lg m$-bit representation of each $x \in S$ into $r - 1$, where $r = n^2$, contiguous parts of size $s = \left\lfloor \frac{\lg m}{n^2} \right\rfloor$ bits and one part (consisting say of the most significant bits of x) of the remaining s' bits, $1 \leq s' \leq s$. We number the parts $0, \ldots, r-1$, with 0 being the most significant. Since $n < \sqrt{\lg m}$, $s \geq 1$, and this is possible. Then there exists a set $R \subseteq [r]$, $|R| = n$, such that if we consider only the bits in the parts that belong to R, all keys in S are still distinct [2]. More precisely, let $h(x, R)$ be the number obtained by extracting the bits in x's representation from parts that belong to R, and concatenating them from most significant to least significant. Then for any distinct $x, y \in S$, $h(x, R) \neq h(y, R)$. Similarly let $q(x, R)$ be the number obtained by extracting the bits in x's representation from parts that do *not* belong to R, and concatenating them from most significant to least significant. The set S is represented as follows.

First, we store an implicit representation of R; this takes $\left\lceil \binom{r}{n} \right\rceil = n \lg n + O(n)$ bits as $r = n^2$. Then, we store $h(x_1, R), h(x_2, R), \ldots, h(x_n, R)$ in that order. Finally, we store $q(x_1, R), q(x_2, R), \ldots, q(x_n, R)$ in that order. Clearly, this representation takes $n \lg m + n \lg n + O(n)$ bits.

To answer $\mathsf{rank}(x)$, we read R first; as $n \lg n = o(\lg m)$ this can be done in $O(1)$ time. Then we compute $h(x, R)$ in $O(1)$ time. We then read $h(x_1, R), \ldots, h(x_n, R)$; since $h(x_i, R)$ is $O((\lg m)/n)$ bits long, all these values can be read in $O(1)$ time. We then find an i such that $h(x_i, R) = h(x, R)$; if such an i exists, we verify the match by reading $q(x_i, R)$ and comparing it with $q(x, R)$, and return $i - 1$ or -1 as appropriate. If such an i does not exist then $x \notin S$. It is easy to see that select can also be supported in constant time using this representation. $\square$

Using the representation of Lemma 5.1 instead of Lemma 2.4 for representing the sets at the bottom level in the proof of Theorem 3.1, we get an indexable dictionary structure that takes $n \lceil \lg m \rceil + o(n)$ bits. One can use this structure to get a result similar to Lemma 3.1, without the additive $O(\lg\lg m)$ term in the space complexity. Using this result instead of Lemma 3.1 in the proof of Theorem 4.1, we get the following.

Theorem 5.1 *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 5.1 *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.*

6 Extensions and applications

In this section, we give some extensions and applications of our succinct indexable dictionary (Theorem 4.1) as well as our fully indexable dictionary for dense sets (Corollary 4.1).

6.1 Multiple Indexable Dictionaries

Here, using our succinct indexable dictionary, we will give a better representation for multiple indexable dictionaries improving on Lemma 3.1.

Let $S_0, S_1, \ldots, S_{s-1}$ all contained in $[m]$ be a given sequence of dictionaries with S_i containing n_i elements, such that $\sum_{i=0}^{s-1} n_i = n$. Note that the representation in Lemma 3.1 of these multiple dictionaries requires an oracle to specify the starting point of each dictionary in the sequence. The representation we develop here does not make use of this assumption, but instead requires that $s = O(n)$. It will be used for k-ary tree representation in the next section. Define the set S as follows:

$$S = \{\langle i, j\rangle : i \in [s], j \in [m] \text{ and } j \in S_i\}.$$

We map the pairs $\langle i, j\rangle, i \in [s], j \in [m]$ to integers in the range $[ms]$ using the obvious mapping $\langle i, j\rangle \mapsto i \cdot m + j$. We represent the n-element set S using our indexable dictionary representation of Theorem 4.1, which takes $\mathcal{B}(n, ms) + o(n) + O(\lg\lg ms)$ bits. As any n-element subset of $[ms]$ corresponds to a unique sequence of s sets (using the inverse of the above mapping), the first term $\mathcal{B}(n, ms)$ is the minimum number of bits required to represent such a sequence of multiple dictionaries.

Now to support the multiple dictionary operations $\mathsf{rank}(x, S_i)$ and $\mathsf{select}(j, S_i)$, we need to find the rank of $\langle i, 0\rangle$ in S even if $\langle i, 0\rangle \notin S$. We can do this by a more detailed inspection of the proof of Theorem 4.1, and potentially modifying S slightly.

If $ms \le 4n\sqrt{\lg n}$, then the set S is dense and so this follows from Lemma 4.1. If $ms > 4n\sqrt{\lg n}$ then we alter m to a new and carefully-chosen value m', and redefine S with the new value of m'; more precisely the pairs in S stay the same, but we change the mapping that takes pairs to integers as $\langle i, j\rangle \mapsto i \cdot m' + j$. By doing this, we ensure that no bucket at the top level of Theorem 4.1 contains elements of the form $\langle x, y\rangle$ and $\langle x', y'\rangle$ for $x \neq x'$ (i.e., all elements in a bucket have the same first co-ordinate). 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.

We now discuss the choice of m'. Recall that if we apply Theorem 4.1 directly to S, we would choose an integer l such that $n\sqrt{\lg n} \le \lfloor ms/2^l \rfloor < 2n\sqrt{\lg n}$ and place x in the bucket $\lfloor x/2^l \rfloor$. Let l be this integer, and let $m' = 2^l \cdot \lceil m/2^l \rceil$, i.e., round the value of m to the next higher multiple of 2^l. Now it is easy to verify that $\lfloor (x \cdot m' + y)/2^l \rfloor \neq \lfloor (x' \cdot m' + y')/2^l \rfloor$ for $x \neq x'$, and thus keys belonging to distinct dictionaries are mapped to different buckets.

However, this increases the universe size to $m's$ from ms. Due to this increase, a direct application of Theorem 4.1 may result in the elements being bucketed according to the mapping $x \mapsto \lfloor x/2^{l'} \rfloor$, for some $l' \ge l$. This issue is most easily dealt with by noting that as $m' < m + 2^l$, $m's < ms(1 + 2^l/m) = ms(1 + \Theta(s/(n\sqrt{\lg n}))) = ms(1 + O(1/\sqrt{\lg n}))$ (recall that $s = O(n)$ by assumption). This in particular means that, for n larger than some constant, $m's < 2ms$, and so retaining the mapping $x \mapsto \lfloor x/2^l \rfloor$ in the proof of Theorem 4.1 gives at most $4n\sqrt{\lg n}$ buckets at the top level, which is immaterial. More importantly, since $m's = ms(1 + O(1/\sqrt{\lg n}))$, the increase in the space is only in the lower-order terms. With this additional power, we now support the multiple dictionary operations as follows:

- To find the size of the set S_i, we do the following. Find the rank of $\langle i+1, 0\rangle$ and the rank of $\langle i, 0\rangle$. The difference gives the size of the set S_i.
- To perform $\mathsf{select}(i, S_j)$, find the rank r of $\langle j, 0\rangle$ and then do $\mathsf{select}(r+i)$ in S. The second coordinate of the element returned by the select operation is the value of the i-th smallest element of S_j.
- To find $\mathsf{rank}(x, S_j)$ find and subtract the rank of $\langle j, 0\rangle$ from $\mathsf{rank}(\langle j, x\rangle)$, if $\mathsf{rank}(\langle j, x\rangle) \geq 0$ and return -1 otherwise.

Thus we have:

Theorem 6.1 *Let $S_0, S_1, \ldots, S_{s-1}$ all contained in $[m]$ be a given sequence of $s = O(n)$ 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 $\mathcal{B}(n, ms) + o(n) + O(\lg\lg m)$ bits and the $\mathsf{rank}(x, S_i)$ and $\mathsf{select}(j, S_i)$ operations can be supported in constant time for any $x \in [m], i \in [s]$ and $j \in \{1, \ldots, n_i\}$. We can also find n_i for each i in constant time. The first term in the space bound is the minimum number of bits required to represent such a sequence of sets.*

6.1.1 Application to Graph Representation

One can use the structure of Theorem 6.1 to represent a graph efficiently. Given a graph G with n vertices, $\{0, 1, \ldots, n-1\}$, and m edges, let $S_i \subseteq [n]$ be the set of all vertices adjacent to vertex i. Now, represent the collection of sets $S_0, \ldots, S_{n-1}$, whose total cardinality is m, using the structure of Theorem 6.1. This requires $m \lg n + o(m) + O(\lg\lg n)$ bits. Now, an adjacency query, i.e., checking whether an edge (i, j) is present in the graph is equivalent to checking whether j belongs to the set S_i, which can be done in constant time using a rank query. Finding the degree of a node i can be supported in constant time by finding the cardinality of the set S_i. This structure has the functionalities of both the adjacency matrix and the adjacency list representations of a graph, while using substantially less space. In addition, it also supports 'random access' in the adjacency list representation.

6.2 Representing k-ary Trees

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 $\mathcal{C}(n, k) = \left\lceil \lg \left(\frac{1}{kn+1} \binom{kn+1}{n} \right) \right\rceil$ [16]. Note that $\mathcal{C}(n, k) \approx n(\lg(k-1) + k \lg \frac{k}{k-1})$ which is close to $n(\lg k + \lg e)$, as k grows. We are interested in finding succinct representations of k-ary cardinal trees that support navigation and other queries in constant time. Building on earlier work by Benoit et al. [5] and Raman and Rao [26], Benoit et al. [6] gave a cardinal tree structure that takes $(\lceil \lg k \rceil + 2)n + o(n) + O(\lg\lg k) = \mathcal{C}(n, k) + \Omega(n)$

bits and answers queries asking for parent, ith child, child with label i, degree and subtree size in constant time.

In this section, we look at a representation of a k-ary cardinal tree that supports the navigational operations in constant time, using our succinct 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 in constant time.

Theorem 6.2 *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 left-to-right numbering of the tree nodes by numbers from $\{0, \ldots, n-1\}$, starting from the root with 0. From now on, we refer the nodes of the tree by these numbers. By a child labeled j of a node x, we mean the child y of x such that the edge (x, y) is labeled j. Let S_x be the set of edge labels out of the vertex x. Then the sets $S_0, \ldots, S_{n-1}$ form a sequence of n sets of total size $n-1$, each being a subset of $[k]$.

Representing these multiple dictionaries using the representation in Theorem 6.1 of the last section, we get a representation for the k-ary tree using at most $\mathcal{B}(n-1, kn) + o(n) + O(\lg\lg m) = \mathcal{B}(n, kn+1) - \lg(kn+1) + o(n) + O(\lg\lg kn) = \mathcal{C}(n,k) + o(n) + O(\lg\lg k)$ bits. By Theorem 6.1, we can support the degree of a node x, the i-th child of a node x, and the ordinal position (the local rank) of the child labeled j, if exists, of a node x, all in constant time. However, the basic navigational operations of going to a child or to the parent are not supported. To support these, we re-examine the proof of Theorem 6.1. Note that in applying Theorem 6.1 to represent our tree, the following set S is stored in an indexable dictionary:

$$S = \{\langle x, j\rangle : x \in [n], j \in [k] \text{ and } \exists \text{ an edge labeled } j \text{ out of node } x\}.$$

The representation supports $\mathsf{rank}(\langle x, j\rangle, S)$ and $\mathsf{select}(\langle x, j\rangle, S)$ in $O(1)$ time. It is easy to verify that:

- $\mathsf{rank}(\langle x, j\rangle, S) + 1$ gives the label of the child labeled j of node x, if it exists, and returns 0 otherwise.
- The first component of $\mathsf{select}(i, S)$ is the parent of the node i. I.e., if the i-th element in S is $\langle x, j\rangle$, then x is the parent of the node i, for $i > 0$.

$\square$

6.3 Multisets

Given a multiset M from $U = [m]$, $|M| = n$, an *indexable multiset* representation for M is a representation for M that supports the following two operations in constant time.

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

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

Sometimes we will also be interested in the following more general version of the rankm operation.

$\mathsf{fullrankm}(x)$: Given $x \in U$, return $|\{y \in M | y < x\}|$.

There is an intimate connection between FIDs and multisets similar to that in Lemma 2.1 as shown below.

Lemma 6.1 *Suppose there is an FID representation for any given set $T \subseteq U$ using $f(|T|, |U|)$ bits of space. Then given a multiset M of n elements from the universe $[m]$, there is a structure to represent M using $f(n, m + n)$ bits of space that supports* $\mathsf{fullrankm}$ *and* $\mathsf{selectm}$ *operations in constant time.*

Proof: Consider the $m + n$ bit representation of M obtained as follows. For $i = 0$ to $m - 1$, represent i by a 1 followed by n_i 0s where n_i is the number of copies of the element i present in the set M. Clearly this representation takes $m + n$ bits since it has m 1s and n 0s.

View this bit sequence as a characteristic vector of a set T of m elements from the universe $[m + n]$. Represent T as an FID using $f(m, m + n) = f(n, m + n)$ bits. It is easy to verify that $\mathsf{fullrankm}(x, M) = \mathsf{select}(x, T) - x + 1$ and $\mathsf{selectm}(i, M) = \mathsf{select}(i, \bar{T}) - i$. (Note that $[m + n]$ starts with element 0.) $\square$

The following corollary follows from Corollary 4.1 and Lemma 6.1.

Corollary 6.1 *Given a multiset M of n elements from the universe $[m]$, there is a structure to represent M and to support* rankm^+ *and* $\mathsf{selectm}$ *operations in constant time using $\mathcal{B}(n, m + n) + o(n)$ bits of space, provided that m is $O(n\sqrt{\lg n})$.*

We now develop an indexable multiset representation (that supports only rankm and $\mathsf{selectm}$ operations) taking $\mathcal{B}(n, m + n) + o(n) + O(\lg \lg m)$ bits for all n. As was alluded to in the introduction (see [12]), the first term is the minimum number of bits required to store such a multiset.

Theorem 6.3 *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.*

Proof: If n is dense in m, i.e. if m is $O(n\sqrt{\lg n})$ then the lemma follows from Corollary 6.1.

If not, then we represent M as follows. First represent the set S of distinct elements present in M using the indexable dictionary structure of Theorem 4.1 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 $n_i - 1$ 0s where n_i is the multiplicity of the element i in M. 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 $\mathsf{rankm}(x, M)$, first find $\mathsf{rank}(x, S)$. If the answer is -1, then return -1. Otherwise $\mathsf{rankm}(x, M)$ is $\mathsf{select}(\mathsf{rank}(x, S)+1, R)$. To find $\mathsf{selectm}(i, M)$, let $r = \mathsf{rank}(i, R)+1$ if $\mathsf{rank}(i, R) \geq 0$ and $r = i - \mathsf{rank}(i, \bar{R})$ otherwise. The value r is precisely the number of 1's up to and including i in the characteristic vector of R. Then $\mathsf{selectm}(i, M) = \mathsf{select}(r, S)$.

To support both $\mathsf{rankm}(x, M)$ and $\mathsf{selectm}(i, M)$ in constant time in this way, 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 Corollary 4.1 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. Clearly this space is $\mathcal{B}(n', m) + \mathcal{B}(n - n', n) + o(n) + O(\lg\lg m)$ which 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 $\binom{m}{n} = \frac{m-n+1}{n}\binom{m}{n-1} \geq 2\binom{m}{n-1}$ 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, $\mathcal{B}(n', m) + n \leq \mathcal{B}(n, m) + n' \leq \mathcal{B}(n, m) + o(n)$. □

6.4 Applications of Succinct Fully Indexable Dense Dictionaries

Here we give more applications of our fully indexable dictionary for dense sets obtained in Corollary 4.1 to dictionaries, multisets and partial sums if we want to support restricted operations.

6.4.1 Set data structure with select

Theorem 6.4 *There is a representation of a set $S \subseteq [m]$ of size n that uses at most $\mathcal{B}(n, m) + o(n)$ bits and supports the* select *in constant time.*

Proof: The proof is essentially as in the proof of Theorem 4.1 except that we store each of the B_i's as a sorted list.

To perform a $\mathsf{select}(i)$ operation, we first perform a $\mathsf{pred}(i)$ at the top level prefix sum representation, to find the bucket B_j in which the i-th element is present. Then a $\mathsf{sum}(j)$ operation at the top level representation gives the prefix sum of the first $j-1$ bucket sizes. Now $i - \mathsf{sum}(j)$ is the rank of the element in the bucket B_j in which we are interested. Since the buckets are sorted, it is easy to find the element of appropriate rank in that bucket. □

As an immediate corollary we get:

Corollary 6.2 *Given a sequence $X = x_1, \ldots, x_n$ of positive integers such that $\sum_{i=1}^{n} x_i = m$, the sequence can be represented using $\mathcal{B}(n,m) + o(n)$ bits to support the partial sum query* $\mathsf{sum}(i, X)$ *in constant time. The first term is the information theoretically minimum number of bits required to represent such a sequence.*

Proof: Consider the set S of partial sum values $S = \{\sum_{j=1}^{i} x_j : 1 \leq i \leq n\}$. As the x_is are positive and add up to m, $|S| = n$ and $S \subseteq [m]$. Represent this set using Theorem 6.4 and observe that $\mathsf{sum}(i, X) = \mathsf{select}(i, S)$. As the mapping from the sequence X to S is invertible, the information theoretic minimum number of bits required to store the partial sum information is $\mathcal{B}(n,m)$. The result follows. □

6.4.2 Multiset with selectm

Theorem 6.5 *Given a multiset M of n elements from $[m]$, there is a representation of M that uses $\mathcal{B}(n, m+n) + o(n)$ bits that supports* selectm *operation in constant time.*

Proof: Use the encoding given in the proof of Lemma 6.1 to convert the multiset M into a set $T \subseteq [m+n]$ of size n. Represent T using the representation of Theorem 6.4 which uses $\mathcal{B}(n, m+n) + o(n)$ bits and supports select operation on T. From the proof of Lemma 6.1, we know that $\mathsf{selectm}(i, M) = \mathsf{select}(i, T) - i$. The theorem follows. □

As an immediate corollary we get:

Corollary 6.3 *Given a sequence $X = x_1, \ldots, x_n$ of non-negative integers such that $\sum_{i=1}^{n} x_i = m$, the sequence can be represented using $\mathcal{B}(n, m+n) + o(n)$ bits to support the partial sum query* $\mathsf{sum}(i, X)$ *in constant time. The first term is the information theoretically minimum number of bits required to represent such a sequence.*

Proof: Consider the multiset M of partial sum values $M = \{\sum_{j=1}^{i} x_j : 1 \leq i \leq n\}$. As the x_is are non-negative and add up to m, $M \subseteq [m]$. Represent this multiset using Theorem 6.5 and observe that $\mathsf{sum}(i, X) = \mathsf{selectm}(i, M)$. Also, as the mapping from X to M is invertible, the information theoretic minimum number of bits required to store the partial sum information is $\mathcal{B}(n, m+n)$. The result follows. □

7 Optimality Considerations

As mentioned in the introduction, some of the space bounds we show above may actually be very far from optimal. Recall that, for example, in the context of storing a set of size n from $[m]$, the information theoretic lower bound of $\mathcal{B}(n,m)$ bits may be dwarfed by additive terms of $o(n)$ bits, say when $n = m - c$ for some constant c. We now note that this is unavoidable to some extent, and in particular that achieving a space bound even polynomial in the information-theoretic lower bound and preserving constant query time is impossible for several of the problems that we consider in this paper, namely:

- Supporting rank queries on a set of integers;
- Supporting select queries on a set of integers and

- Supporting $\textsf{sum}$ queries on a sequence of non-negative integers (or equivalently, supporting $\textsf{select}$ on a multiset of integers).

We show that the predecessor problem reduces to all of these. Given a set S of size n from $U = [m]$, recall that the predecessor of x in S for any $x \in U$ is defined as $\max\{y \in S | y < x\}$, and the predecessor problem is to store S to support predecessor queries on S. Beame and Fich showed that on the cell probe model with word size $(\lg m)^{O(1)}$, any data structure that used $n^{O(1)}$ words of space would require $\Omega(\sqrt{\lg n / \lg\lg n})$ time to solve the predecessor problem, for some range of n relative to m [4]. It is not hard to see that the same lower bound applies to supporting $\textsf{fullrank}$ queries: simply take any data structure for $\textsf{fullrank}$ that purports to do better than the predecessor lower bound, and augment it with an array of n words that contains the elements of S in sorted order. Using this array, one can translate the answer to a $\textsf{fullrank}(x, S)$ query into the actual predecessor of x in S in $O(1)$ further time, thus violating the lower bound for the predecessor problem.

Lemma 7.1 *Given a set $S \subseteq [m], |S| = n$, one cannot support* $\textsf{rank}$ *queries on S in $O(1)$ time for all n using $(\mathcal{B}(n, m))^{O(1)}$ bits in the cell probe model with word size $(\lg m)^{O(1)}$ bits.*

Proof: Suppose by way of contradiction that the statement of the lemma is false. Then we would solve $\textsf{fullrank}$ queries on S in $O(1)$ time using $n^{O(1)}$ words of space as follows. Letting $\mathcal{B} = \mathcal{B}(n, m)$, by assumption we can store S in $\mathcal{B}^{O(1)}$ bits, which is $n^{O(1)}$ words of space, and answer $\textsf{rank}$ queries on S in $O(1)$ time. Similarly, we can store $\bar{S}$ in $\mathcal{B}(m - n, m) = n^{O(1)}$ words of space and answer $\textsf{rank}$ queries on $\bar{S}$ in $O(1)$ time. To answer $\textsf{fullrank}(x, S)$ for all $x \in U$, we first compute $\textsf{rank}(x, S)$; if the value returned is not -1 we return it as the answer to $\textsf{fullrank}$, otherwise we return $x - \textsf{rank}(x, \bar{S}) - 1$ as the answer to $\textsf{fullrank}$. □

Lemma 7.2 *Given a set $S \subseteq [m], |S| = n$, one cannot support* $\textsf{select}$ *queries in $O(1)$ time for all n using $(\mathcal{B}(n, m))^{O(1)}$ bits in the cell probe model with word size $(\lg m)^{O(1)}$ bits.*

Proof: Suppose by way of contradiction that the statement of the lemma is false. Then given $T \subseteq [m^*]$, where $T = \{t_1, \ldots, t_{n^*}\}$ and $t_1 < t_2 < \ldots < t_{n^*}$, we can answer predecessor queries on T in $O(1)$ time using $(n^*)^{O(1)}$ words as follows. We create a bit-vector by writing down t_1 0s followed by a 1, then $(t_2 - t_1)$ 0s followed by a 1, and so on and finally we write $(m^* - t_n)$ 0s. This is a bit vector with n^* 1s and m^* 0s; we view this is as a characteristic vector of a set S of size $n = n^*$ from $[m]$ where $m = m^* + n^*$. We store $\bar{S}$ using $(\mathcal{B}(n, m))^{O(1)}$ bits $= n^{O(1)}$ words and compute $\textsf{fullrank}(x, T)$ as $\textsf{select}(x, \bar{S}) - x$ in $O(1)$ time. □

Lemma 7.3 *Given a sequence $X = x_1, x_2, \ldots, x_n$ of non-negative integers adding to m, we cannot store this sequence in $(\mathcal{B}(n, m+n))^{O(1)}$ bits of space for all m, n and support the* $\textsf{sum}$ *query on this sequence in $O(1)$ time on the cell probe model with word size $(\lg(m + n))^{O(1)}$ bits.*

Proof: Suppose by way of contradiction that the statement of the lemma is false. Then given $S = \{s_1, \ldots, s_{n^*}\} \subseteq [m^*]$ where $s_1 < s_2 < \ldots < s_{n^*}$ we could answer predecessor queries on S in $O(1)$ time using $(n^*)^{O(1)}$ words of space, as follows. Create a sequence X

which consists of s_1 0s followed by a 1, and for $i = 2$ to n, $s_i - s_{i-1} - 1$ 0s followed by a 1 and finally $m^* - s_{n^*} - 1$ 0s followed by a 1. This is a sequence of $n = m^* + 1$ non-negative integers adding to $m = n^*$. We store this sequence using $(\mathcal{B}(n, m+n))^{O(1)}$ bits, but since $\mathcal{B}(n, m+n) = \mathcal{B}(m, m+n) = \mathcal{B}(n^*, m^*+n^*+1) \leq \mathcal{B}(n^*, 2m^*+1)$, the space usage is $(n^*)^{O(1)}$ words. It is easy to verify that $\mathsf{fullrank}(j, S) = \mathsf{sum}(j, X)$. $\square$

8 Conclusions

We have given a static dictionary structure to store an n element subset of an m element universe, that takes $\mathcal{B}(n, m) + o(n) + O(\lg\lg m)$ bits of space and supports rank 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 support 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 a succinct representation for an indexable 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 $\mathcal{B}(n, m) + o(n)$ bits 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.

Some open problems that remain are:

1. Is there a succinct indexable dictionary taking $\mathcal{B}(n, m) + o(n)$ bits in the RAM model?

2. Is there a representation for k-ary trees taking $\mathcal{C}(n, k) + o(n) + O(\lg\lg k)$ bits that can also support subtree size operation besides the other navigational operations in constant time?

3. The lower order terms in the space complexity of our succinct dictionary and prefix sums representation are $o(\mathcal{B})$, where $\mathcal{B}$ is the information theoretically minimum space, only for certain values of n and m. We argue in Section 7 that this cannot be extended for all values of n and m. It is easy to determine the ranges for which our representations are within $o(\mathcal{B})$ of optimal. It will be nice to see whether these ranges can be extended, if not completely (which we know we can't).

Acknowledgments. We thank the anonymous referee for helpful comments.

References

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

[2] M. Ajtai, M. L. Fredman, and J. Komlós. Hash functions for priority queues. *Information and Control*, 63(3):217–225, 1984.

[3] A. Andersson, T. Hagerup, S. Nilsson, and R. Raman. Sorting in linear time? *Journal of Computer and System Sciences*, 57:74–93, 1998.

[4] P. Beame and F. E. Fich. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38–72, 2002.

[5] D. Benoit, E. D. Demaine, J. I. Munro, and V. Raman. Representing trees of higher degree. In *Proceedings of Workshop on Algorithms and Data Structures*, volume 1663 of *LNCS*, pages 169–180. Springer, 1999.

[6] D. Benoit, E. D. Demaine, J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Representing trees of higher degree. To appear in *Algorithmica*. Preliminary version in University of Leicester Technical Report 2001/46, 2001.

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

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

[9] 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.

[10] J. G. Cleary. Compact hash tables using bidirectional linear probing. *IEEE Transactions on Computers*, 33:828–834, 1984.

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

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

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

[14] 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, 1984.

[15] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47:424–436, 1993.

[16] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. Concrete Mathematics. Addison-Wesley, 1989.

[17] R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. Manuscript, 2002. Preliminary version in *Proceedings of the ACM Symposium on Theory of Computing*, pages 397–406. ACM Press, 2000.

[18] T. Hagerup and T. Tholey. Efficient minimal perfect hashing in nearly minimal space. In *Proceedings of Symposium on Theoretical Aspects of Computer Science*, volume 2010 of *LNCS*, pp. 317–326. Springer, 2001.

[19] T. Hagerup. Sorting and searching on the word RAM. In *Proceedings of Symposium on Theoretical Aspects of Computer Science*, volume 1373 of *LNCS*, pages 366–398. Springer, 1998.

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

[21] J. I. Munro. Tables. In *Proceedings of Foundations of Software Technology & Theoretical Computer Science*, volume 1180 of *LNCS*, pages 37–42. Springer, 1996.

[22] J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, 2001.

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

[24] R. Pagh. Low redundancy in static dictionaries with constant query time. In *SIAM Journal on Computing*, 31(2):353-363, 2001.

[25] W. J. Paul and J. Simon. Decision trees and random access machines. In *Proceedings of International Symposium on Logic and Algorithmic*, Zürich, pp. 331–340, 1980.

[26] V. Raman and S. S. Rao. Static dictionaries supporting rank. In *Proceedings of International Symposium on Algorithms and Computation*, volume 1741 of *LNCS*, pages 18–26. Springer, 1999.

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

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

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