On the Succinct Representation of Equivalence Classes

by

Hicham El-Zein

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2014

Abstract

Given a set of n elements that are partitioned into equivalence classes, we study the problem of assigning unique labels to these elements in order to support the query that asks whether the elements corresponding to two given labels belong to the same equivalence class. This problem has been studied by Katz et al. [15], Alstrup et al. [2], and Lewenstein et al. [16]. Lewenstein et al. [16] showed that with no auxiliary data structure, a label space of size $\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor$ is necessary and sufficient to represent the equivalence relation. They also showed that if the labels were to be assigned from the set $\{1, \ldots, n\}$, a data structure of $\Theta(\sqrt{n})$ bits is necessary and sufficient to represent the equivalence relation and to answer the equivalence query in $\Theta(\lg n)$ time. In this thesis, we give an improved data structure that uses $\Theta(\sqrt{n})$ bits and can answer queries in $\Theta(1)$ time, when the label space is of size n. Moreover, we study the case where we allow the label space to be of size cn for any constant $c > 1$. We show that with such a label space, a data structure of $\Theta(\lg n)$ bits is necessary and sufficient to represent the equivalence relation and to answer the equivalence query in constant time. We believe that our work can trigger further work on tradeoffs between label space and auxiliary data structure space for other labeling problems.

Acknowledgements

I offer my sincere gratitude to my supervisor, Professor J. Ian Munro, who has supported me throughout my research with his patience and knowledge while allowing me to work on topics that I like.

I also wish to thank Professor Venkatesh Raman for the insightful discussions that we had.

Last, but not least, I would like to thank my readers, Professor Timothy Chan and Professor Alejandro Lòpez-Ortiz. Moreover, I would like to highlight Professor Timothy Chan's invaluable comments that have greatly improved this thesis.

Dedication

This is dedicated to my parents and family.

Table of Contents

Chapter 1

Introduction and Motivation

Given a partition of an n-element set into equivalence classes, our problem is to preprocess the set, assigning a unique label to each element, to obtain a data structure with minimum space to quickly support the following query: given two labels, determine whether their corresponding elements are in the same equivalence class. We call such queries 'equivalence queries'. This is a fundamental data structures problem and it has various applications such as testing whether two vertices are in the same connected component in an undirected graph or in the same strongly connected component in a directed graph. We study the problem from the perspective of succinct data structures. Our aim is to develop data structures whose size is within a constant factor of the information theoretic lower bound. Designing succinct data structures is an area of interest in theory and practice motivated by the need for storing large amount of data using the smallest space possible. For succinct representations of dictionaries, trees, arbitrary graphs and partially ordered sets see [5, 20, 11, 19, 4, 10, 8, 3, 7, 18].

Kannan, Naor and Rudich [14] were the first to introduce the concept of labeling schemes to answer graph adjacency queries. Katz, Katz, Korman and Peleg [15] extended this notion to support graph flow and connectivity queries. They studied the problem of assigning (not necessarily distinct) labels to graph nodes so that queries can be answered by just looking at the labels of the queried nodes. They showed that to answer k-connectivity queries the length of each label has a lower bound of $\Omega(k \lg n)$ bits. For the problem that is considered in this thesis $k = 1$, their lower bound implies a $\lceil \lg n \rceil$ lower bound for the length of each label. However, in some situations we may want to distinguish between individual nodes within the same component, so we may want to give unique labels to each node. Lewenstein et al. [16] studied this problem in two models:

- In the first model, the query is to be answered by only examining the labels of the queried elements without having any auxiliary information stored other than the value of n. They called this problem the *direct equivalence queries* problem. They tightened the lower bound in [2] to a label space of $\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor$, which can be represented in $\lg n + \lg\lg n - \Omega(1)$ bits, and proved that it is sufficient. Moreover, they showed that one can solve the problem using $\lg n + \lg\lg n + 2$ bits such that equivalence queries can be answered in $\Theta(1)$ time.

- In the second model, the n-element s are to be assigned unique labels from the set $\{1, \ldots, n\}$. They showed that an auxiliary data structure of $\Theta(\sqrt{n})$ bits is necessary and sufficient to represent the equivalence class information. They supported the query in such a structure in $\Theta(\lg n)$ time. Moreover, they developed structures where queries can be answered:
  - in $O(\lg\lg n)$ time using $O(\sqrt{n} \lg n/\lg\lg n)$ bits, and
  - in $O(1)$ time using $O(\sqrt{n} \lg n)$ bits of space.

In this thesis, we give an improved data structure that uses $\Theta(\sqrt{n})$ bits and can answer queries in $\Theta(1)$ time, when the label space is of size n. Moreover, we notice an inversely proportional relation between label space and auxiliary data structure size. If the label space is in the range of $\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor$, no data structure is required. Once the label space range is decreased to n, a data structure of size $O(\sqrt{n})$ bits is necessary. We further investigate this relation. Our work is motivated by the fact that unless n is a power of two or a little less than a power of two, we can increase the label space by a constant factor without increasing the number of bits required to store each address. To be more specific, we investigate the case where the labels are to be assigned from the set $\{1, \ldots, cn\}$ where $c > 1$ (c can be a real value less than 2). We show that an auxiliary data structure whose size is $\Theta(\lg n)$ bits is necessary and sufficient to represent the equivalence class information in this case. Such a structure can be completely stored in cache memory. Moreover, we can support the query in such a structure in $\Theta(1)$ time.

This thesis is divided as follows. In chapter 2 we cover the succinct representation of bit vectors. In chapter 3, we cover the direct equivalence query problems. In chapter 4, we revise the main data structure of [16] and provide a data structure with improved query time. In chapter 5 we present our data structure to represent the equivalence class information using $O(\lg n)$ bits in the model where the label space is cn for any $c > 1$. Moreover, we give a data structure that represent the equivalence class information using $O(\lg n/\lg f(n))$ bits in the model where the label space is $f(n) \cdot n$ where $f(n) \in \omega(1)$ and

$f(n) \in O(\lg n)$. In chapter 6, we show that $\Omega(\lg n)$ bits are necessary when the label space is cn. Finally, we conclude our work in chapter 7.

1.1 Computational Model

The computational model used in this thesis is the word random access machine model, or the word RAM model [1, 9]. The word RAM model is a realistic and natural model for describing modern computers. Data is stored in words consisting of $w \in \Omega(\lg n)$ bits, where n is the input size. Words can be read and written in constant time. Moreover, arithmetic (addition, subtraction, multiplication, and division) and bitwise boolean operations (AND, OR, NOT, XOR, SHIFT etc.) can be done in $O(1)$ time on a constant number of words. We measure the running time of an algorithm in this model by counting the number of memory accesses and operations performed on words. The space cost can be measured by counting the number of words or the number bits used by the algorithm.

1.2 Definitions

In this section, we briefly describe the required definitions and background. An *integer partition* p of n is a multiset of positive integers that sum to n. We call these positive integers the *class sizes* of p, and we denote by $|p|$ this number of classes. We say that a partition p of n *dominates* a partition q of m where $n > m$ if q *fits* in p. To be more precise p dominates q if:

- $|p| \geq |q|$ and
- for $1 \leq i \leq |q|$, the i^{th} largest class size (breaking ties arbitrarily) in p is at least as big as the i^{th} largest class size in q.

For example, the partition $\{7, 7, 6\}$ of 20 dominates the partition $\{5, 5\}$ of 10, but not the partition $\{8, 2\}$ of 10. Given a partition p of n, we define a *part* q of size k to be a collection of class sizes in p that sum to k. We say that a size s *fills* q if q contains $\lfloor k/s \rfloor$ classes of size s and a class of size $k \bmod s$. Finally we say that two parts *intersect* if they share at least one common class; otherwise, they are *non-intersecting*. For example the partition $\{1, 4, 5\}$ of 10 has the following parts:

- Part $\{1\}$ of size 1.

- Part $\{4\}$ of size 4.
- Part $\{5\}$ of size 5.
- Part $\{1, 4\}$ of size 5.
- Part $\{1, 5\}$ of size 6.
- Part $\{4, 5\}$ of size 9.
- Part $\{1, 4, 5\}$ of size 10.

We say that 5 fills the part $\{4, 5\}$. The parts $\{4, 5\}$ and $\{4\}$ are intersecting, while the parts $\{4, 5\}$ and $\{1\}$ are non-intersecting.

Chapter 2

Bit Vectors

For the sake of completeness and since bit vectors are used extensively throughout this thesis, we cover them in this chapter. A bit vector is a simple way to represent a set S from the universe $[m]$, where $[m]$ denotes the set $\{0, 1, \ldots, m-1\}$. If $i \in S$, we set the i^{th} bit in the bit vector to 1, otherwise we set it to 0. It is not hard to see that membership queries (checking whether a given element in $[m]$ belongs to S) can be answered in constant time by probing a single bit. In addition to membership queries, we would also like to support the following operations:

- ***rank(i)***: returns the number of 1s up to position i.
- ***select(i)***: return the position of the i^{th} 1.

Given a bit vector of length m, Jacobson [13] gave a structure that takes $o(m)$ additional bits of space and can support rank and select by making $O(\lg m)$ bit inspections. However, the bits inspected were not necessarily contiguous and might depend on previous values read. Munro [17] enhanced this structure to support both operations in constant time, without increasing the space bound. In this chapter, we describe the details of this structure.

Supporting Rank. To answer rank queries in constant time, we store the following:

- Break the vector into blocks of size $\lceil \lg^2 m \rceil$, and store in a table T_1 the number of $1s$ up to the last position of each block. Also, store in T_1 references to all tables T_{2i} (described below) where $0 \leq i \leq \lceil n/\lceil \lg^2 m \rceil \rceil$. T_1 requires $O(m/\lg m)$ bits.

- Break the blocks into sub-blocks of size $\frac{1}{2}\lceil\lg m\rceil$, and for each block i store in a table T_{2i} the number of 1s from the start position of the block up to the last position of each sub-block. T_{2i} requires $O(\lg m \lg\lg m)$ bits. The total space required by all such tables is $O(m \lg\lg m/\lg m)$ bits.

- For every possible sub-block, store a table T_3 that gives the number of 1s up-to every possible position. Since there is $O(\sqrt{m})$ distinct sub-blocks, T_3 requires $O(\sqrt{m}\lg m \lg\lg m)$ bits.

To answer *rank(x)*, Let $i = \lfloor x/\lceil\lg^2 m\rceil\rfloor$ be the index of the block containing x, compute j_1 the number of ones up to position $(i \cdot \lceil\lg^2 m\rceil)$ using table lookup on T_1. Let $k = \lfloor(x - i \cdot \lceil\lg^2 m\rceil)/(\frac{1}{2}\lceil\lg m\rceil)\rfloor$ be the index of the sub-block containing x, using table lookup on T_{2i} compute j_2 the number of ones up to the last position in the $(k-1)$-th sub-block of the i^{th} block of S. Finally using table lookup on T_3, get j_3 the number of ones up to position $(x - i \cdot \lceil\lg^2 m\rceil - k \cdot \frac{1}{2}\lceil\lg m\rceil)$ in the k^{th} sub-block of the i^{th} block of S, and return $(j_1 + j_2 + j_3)$.

Supporting Select. Supporting select queries is more complex than supporting rank queries. We store the following:

- In a table T_1, store the position of every $\lceil\lg m \lg\lg m\rceil$)-th 1 bit in the bit vector. Also, store in T_1 references to all tables T_{2i} (described below) where $0 \leq i \leq \lceil n/\lceil\lg m \lg\lg m\rceil\rceil$. T_1 requires $O(m/\lg\lg m)$ bits.

- Let r be the sub-range between the i^{th} 1 and the $(i+1)^{th}$ 1 in T_1. If $r \geq \lceil\lg m \lg\lg m\rceil^2$, store all the positions of all ones in this subrange in the table T_{2i}. In this case, T_{2i} would require $O(\lg^2 m \lg\lg m)$ bits. However there can be at most $m/\lceil\lg m \lg\lg m\rceil^2$ such sub-ranges. Thus the total space required by such tables would be $O(m/\lg\lg m)$ bits. If $r < \lceil\lg m \lg\lg m\rceil^2$ store the position of every $\lceil\lg r \lg\lg m\rceil$-th one bit in the sub-range. In this case, T_{2i} would require $O(r/\lg\lg m)$ bits, and the total space of such tables is $O(m/\lg\lg m)$ bits.

- After one more level of subdivision, the range size will be at most $(\lg\lg m)^4$. We use a precomputed table T_3 that requires $o(m)$ bits to store answers of all select queries on every possible bit vector of that size.

To answer *select(x)*, we check if x is a multiple of $\lceil\lg m \lg\lg m\rceil$. If so we can answer *select*(x) using table lookup on T_1. Let $i = \lfloor x/\lceil\lg m \lg\lg m\rceil\rfloor$. Using table lookup on T_1, get j_1 the index of the $(i \cdot \lceil\lg m \lg\lg m\rceil)$-th one in S and j_2 the index of the $((i+1) \cdot$

$\lceil \lg m \lg \lg m \rceil$)-th one in S. If $r = j_2 - j_1 \geq \lceil \lg m \lg \lg m \rceil^2$ we get j_3 the index of the $(x - i \cdot \lceil \lg m \lg \lg m \rceil)$-th one in the subdivision between j_1 and j_2 using table lookup on T_{2i}, and we return $(j_1 + j_3)$. Let $k = \lfloor (x - i \cdot \lceil \lg m \lg \lg m \rceil)/\lceil \lg r \lg \lg m \rceil \rfloor$. Using table lookup on T_{2i}, get j_4 the index of the $(k \cdot \lceil \lg r \lg \lg m \rceil)$-th one in the subdivision between j_1 and j_2, and j_5 the index of the $((k+1) \cdot \lceil \lg r \lg \lg m \rceil)$-th one in the subdivision between j_1 and j_2. Finally using table lookup on T_3, we get j_6 the index of the $(x - i \cdot \lceil \lg m \lg \lg m \rceil - k \cdot \lceil \lg r \lg \lg m \rceil)$-th one in the subdivision between j_4 and j_5, and return $(j_1 + j_4 + j_6)$.

An immediate use of rank and select queries, is the ability to support the successor and predecessor queries.

Supporting Predecessor. The predecessor of an element x, is the largest element $y \leq x$ such that $y \in S$. To answer $\mathit{predecessor}(x)$ return $\mathit{select}(\mathit{rank}(x))$.

Supporting Successor. The successor of an element x, is the smallest element $y \geq x$ such that $y \in S$. To answer $\mathit{successor}(x)$ we check if $x \in S$ if so we return x, otherwise we return $\mathit{select}(\mathit{rank}(x) + 1)$.

Theorem 1. ([17]) *A bit vector of length m can be represented in $m + o(m)$ bits, such that rank, select, membership, predecessor and successor queries can be answered in constant time.*

Chapter 3

Direct Equivalence Query Problems

This chapter covers labeling schemes where elements are to be given unique labels, and the equivalence query is to be answered by computing from the two labels without using any auxiliary data except for the value of the number of elements n. We give tight bounds on this problem in both the static and dynamic models. Storing n requires $\lceil \lg n \rceil$ bits. However, note that we can round n to the nearest power of 2 and store it in $\lceil \lg \lg n \rceil$ bits.

3.1 Direct Equivalence Query Problem

In the direct equivalence query problem, we give each element a unique label such that we can answer the equivalence query by computing directly from the two labels without using any auxiliary data except for the value of n. This problem was studied by Alstrup et al. [2], where they showed that $\lg n + \Theta(\lg \lg n)$ bits of space are necessary and sufficient to represent the labels. This bound was further strengthened by Lewenstein et al. [16] to $\lg n + \lg \lg n + \Theta(1)$). In this section we review their results:

Theorem 2. ([16]) *Let a partition of an n-element set into equivalence classes be given as input to the direct equivalence query problem. Then a label space of $\sum_{i=1}^{n} \lfloor n/i \rfloor$ is necessary and sufficient to represent the labels.*

Proof. Order the classes in decreasing order by their sizes. The key observation is that the i^{th} class contains at most $\lfloor n/i \rfloor$ elements. For the upper bound, simply assign labels from the set of integers in the range of $\sum_{j=1}^{i-1} (\lfloor n/i \rfloor) + 1$ to $\sum_{j=1}^{i} \lfloor n/i \rfloor$ for the i^{th} class.

To show that this many labels are necessary, consider the collection C of n equivalence relations. The relation C_i contains i classes of size $\lfloor n/i \rfloor$ and one class of size $n - i \cdot \lfloor n/i \rfloor$.

Consider the labels assigned by any labeling scheme for the above collection of equivalence relations. The labels assigned to the relation C_1 can be assigned to at most one class of each of C_i, where $2 \leq i \leq n$. The reason for this is that every pair of elements in C_1 are in the same equivalence class, so a conflict will occur if these labels were assigned to more than one class of C_i. Now remove C_1 and all the classes from C_i that have been assigned the same labels as C_1. Repeat the same argument with the labels assigned to C_2, C_3, up to C_n in that order. $\square$

To answer the equivalence query in the above labeling scheme, given an integer label x, we need to find the largest i such that $\sum_{j=1}^{i} \lfloor n/i \rfloor < x$. In order to support this query in constant time, we slightly increase the address space. Order the classes in non-increasing order by their sizes, and give the i^{th} class label i. Order the elements within each class arbitrarily. The label for an element x is given by a pair (i, j), where i is the label of the class to which x belongs to, and j is the rank of x in class i. Since the size of i is not fixed, prefix the label (i, j) by storing the length of i (i.e. $\lceil \lg i \rceil$) in binary using $\lceil \lg \lceil \lg n \rceil \rceil$ bits. The equivalence query can be answered in constant time by checking i. The number of bits used per label is $\lceil \lg \lceil \lg n \rceil \rceil + \lceil \lg i \rceil + \lceil \lg \lfloor n/i \rfloor \rceil$ which is at most $\lg n + \lg \lg n + 2$.

Theorem 3. ([16]) *Let a partition of an n-element set into equivalence classes be given as input to the direct equivalence query problem. Then $\lg n + \lg \lg n - \Theta(1)$ bits are necessary and sufficient to represent each of the labels. Moreover by using a label space of $\lg n + \lg \lg n + 2$ bits the equivalence query can be answered in $\Theta(1)$ time.*

3.2 Online Direct Equivalence Query Problem

In the problem, which we call the *online direct equivalence query* problem, each element is to be given a unique label, and the equivalence query is to be answered by computing directly from the two labels without using any auxiliary data except for the value of the number of elements n. In contrast to the static setting, only n is known in advance, the labeling scheme receives the partition as an online sequence of events, where each event contain a class of the partition and its elements. Moreover, re-labeling elements is not allowed.

This problem was studied by Dahlgaard et al. in [6]. In this section, we review their results; however we provide alternative proofs.

Theorem 4. ([6]) *Let a partition of an n-element set into equivalence classes be given as input to the online direct equivalence query problem. Then a label space of $\sum_{i=1}^{n} i = n(n+1)/2$ is necessary and sufficient to represent the labels.*

Proof. Order the classes in chronological order as they are received by the algorithm. Our key observation is that the i^{th} class contains at most $n-i$ elements. For the upper bound, simply assign label 1 for the last class ($i = n$) and assign labels from the set of integers $(\sum_{j=1}^{n-i} j) + 1$ to $\sum_{j=1}^{n-i+1} j$ for the i^{th} class where $1 \leq i < n$.

To show that this many labels are necessary, consider the collection C of n equivalence relations. The relation C_1 contains one class with n-element s, and for $1 < i \leq n$ relation C_i contains i classes such that classes 1 till $i-1$ contain only one element each, and class i contains $n-i$ elements.

Consider the labels assigned by any labeling scheme for the above collection of equivalence relations. C_1 requires n distinct labels. The first element in C_i, where $2 \leq i \leq n$ will be assigned the same label as the first element in C_1. However, the remaining $n-1$ labels assigned to C_1 cannot be assigned to any element in C_i or a conflict will occur. Now remove C_1 and the first elements of C_i. Repeat the same argument with the labels assigned to C_2, C_3, up to C_n in that order. □

To answer the equivalence query in the above labeling scheme, given an integer label x, we need to find the largest i such that $\sum_{j=1}^{n-i} j < x$. Notice that

$$(n-i)(n-i+1)/2 < x \leq (n-i+1)(n-i+2)/2$$

and

$$(n-i)(n-i+1) < 2x \leq (n-i+1)(n-i+2)$$

and

$$\sqrt{(n-i)(n-i+1)} < \sqrt{2x} \leq \sqrt{(n-i+1)(n-i+2)}.$$

Thus,

$$(n-i) \leq \lfloor \sqrt{2x} \rfloor < (n-i+2)$$

so $i = n - \lfloor\sqrt{2x}\rfloor$ or $i = n + 1 - \lfloor\sqrt{2x}\rfloor$. We can check both cases in constant time, so the query time is equivalent to the time needed to compute $\lfloor\sqrt{2x}\rfloor$ which can be done in $O(\lg\lg n)$ time using Newton's iterative method.

In order to support this query in constant time, we modify the labeling scheme so that all classes have size equal to n. Given two integer labels x and y, they belong to the same equivalence class if and only if $\lfloor x/n\rfloor = \lfloor y/n\rfloor$.

Theorem 5. *Let a partition of an n-element set into equivalence classes be given as input to the online direct equivalence query problem. Then $\lceil 2\lg n\rceil - 1$ bits are necessary and sufficient to represent each of the labels and answer the equivalence query in $\Theta(\lg\lg n)$ time. Moreover using a label space of $\lceil 2\lg n\rceil$ bits the equivalence query can be answered in $\Theta(1)$ time.*

Chapter 4

Data Structures with Label Space n

In this chapter, we revise the main data structure of [16] and we give correction to lemma 2 and lemma 3 in [16]. Then, we provide an improved data structure using $O(\sqrt{n})$ bits that answers queries in constant time.

We assume that the equivalence class is given by a tuple containing the sizes of the classes, and our task is to give each element a unique label from the range 1 to n. We can use an auxiliary data structure to answer the equivalence query. First we assign an implicit ordering of the elements, where each element gets a label according to this ordering. Then, given two labels the equivlance query is answered by looking at these two labels and the augmented data structure.

The number of partitions of an n-element set into equivalence classes is the same as the number of integer partitions of n, which by the Hardy-Ramanujan formula [12] is asymptotically equivalent to $\frac{1}{4n\sqrt{3}}e^{\pi\sqrt{\frac{2n}{3}}}$. Thus the information theoretic lower bound for representing the equivalence class relation is $\Theta(\sqrt{n})$ bits of space.

Let k be the number of distinct class sizes. For $i = 1$ to k, let s_i be the distinct sizes of the classes, and let n_i be the number of classes of size s_i. Order the classes in non-decreasing order by $\gamma_i = s_i n_i$ so that for $i = 1$ to $k-1$, $s_i n_i \leq s_{i+1} n_{i+1}$. Notice that since

$$\textstyle\sum_{i=1}^{k} s_i n_i = n \text{ and } s_i n_i \geq i \text{ for } i = 1, \ldots, k,$$

k is at most $\sqrt{2n}$.

4.1 Succinct Structure Using $O(\sqrt{n})$ bits

In this section, we design a data structure using $O(\sqrt{n})$ bits of space to represent the equivalence class information and support the equivalence query in $O(\lg n)$ time. The primary data structure is made up of two sequences:

- the sequence $\vec{\delta}$ that consists of $\delta_1 = s_1 n_1$ and $\delta_i = s_i n_i - s_{i-1} n_{i-1}$, for $i = 2, \ldots, k$ and
- the sequence $\vec{n}$ that consists of n_i, for $i = 1, \ldots, k$.

Elements of the two sequences are represented in binary. Since the length of each element may vary, we store two other sequences that shadow the primary sequences. The shadow sequences have a 1 at the starting point of each element in the shadowed sequence and a 0 elsewhere. Also store a select structure (see Chapter 2) on the two shadow sequences in order to identify the 1s quickly.

Lemma 6. ([16]) *For any integer i where $1 \le i \le n$, $|\{\delta_j \mid \delta_j \ge i\}| \le \sqrt{2n/i}$.*

Proof. Let $\delta_{j_t} \ge i$ and $j_t < j_{t+1}$, for $t = 1, \ldots, b$. Then $s_{j_t} n_{j_t} \ge ti$. Since

$$\textstyle\sum_{t=1}^{b} ti \le \sum_{t=1}^{b} s_{j_t} n_{j_t} \le n,$$

we have $b(b+1)/2 \le n/i$ and $b \le \sqrt{2n/i}$, so our claim holds. □

Lemma 7. ([16]) $\sum_{j=1}^{k} \lg(\max\{1, \delta_j\}) \in O(\sqrt{n})$.

Proof. Break the δ values into ranges of powers of two. Then

$$\textstyle\sum_{j=1}^{k} \lg(\max\{1, \delta_j\}) \le \sum_{p=1}^{\lceil \lg n \rceil} p\sqrt{2n/2^{p-1}} = 2\sqrt{n} \sum_{p=1}^{\lceil \lg n \rceil} \frac{p}{2^{p/2}} \in O(\sqrt{n}).$$

□

Similarly we prove that:

Lemma 8. ([16]) *For any integer i where $1 \le i \le n$, $|\{n_j \mid n_j \ge i\}| \le \sqrt{2n/i}$.*

Proof. Let $n_{j_t} \ge i$ and $s_{j_t} < s_{j_{t+1}}$, for $t = 1, \ldots, b$, then $s_{j_t} n_{j_t} \ge ti$. Since

$$\sum_{t=1}^{b} ti \leq \sum_{t=1}^{b} s_{j_t} n_{j_t} \leq n$$

we have that $b(b+1)/2 \leq n/i$ and $b \leq \sqrt{2n/i}$, so our claim holds. $\square$

Lemma 9. ([16]) $\sum_{j=1}^{k} \lg(n_i) \in O(\sqrt{n})$.

Proof. Break the n values into ranges of powers of two. Then

$$\sum_{j=1}^{k} \lg n_j \leq \sum_{p=1}^{\lceil \lg n \rceil} p\sqrt{2n/2^{p-1}} = 2\sqrt{n} \sum_{p=1}^{\lceil \lg n \rceil} \frac{p}{2^{p/2}} \in O(\sqrt{n}).$$

$\square$

Thus the space occupied by these sequences is $O(\sqrt{n})$. Assign labels to elements based on the first sequence. To be more specific, for the classes of size s_i, assign label values from $\sum_{j=1}^{i-1} s_j n_j + 1$ to $\sum_{j=1}^{i} s_j n_j$. Next, we show how to implement the equivalence query. Given an element labeled x, first find the predecessor $p(x)$ of x, which is $\max\{j \mid \sum_{i=1}^{j} s_i n_i < x\}$. Given x and y, if $p(x) \neq p(y)$ then x and y belong to different equivalence classes. If $p(x) = p(y)$, then we know that x and y belong to equivalence classes of the same size. They are in the same equivalence class if and only if $\lceil (x - \sum_{i=1}^{p(x)} s_i n_i)/n_{p(x)+1} \rceil$ is equal to $\lceil (y - \sum_{i=1}^{p(y)} s_i n_i)/n_{p(y)+1} \rceil$. Using the select data structures saved we can find the value of n_i in constant time, so what is left is answering the predecessor query.

To answer the predecessor query efficiently, store the partial sum values $\sum_{j=1}^{i} s_j n_j$ for each i that is a multiple of $\lg n$. Now, an $O(\lg n)$ range of the predecessor can be obtained by doing a binary search for x on these stored partial sums. Once this range is found, the actual predecessor can be computed by doing a linear search on the δ values in this range. Since both operations take $O(\lg n)$ time, we obtain the following theorem:

Theorem 10. ([16]) *Given a partition of an n-element set into equivalence classes $O(\sqrt{n})$ bits are necessary and sufficient for storing the partition and to answer the equivalence query in $O(\lg n)$ time if each element is to be given a unique label in the range $\{1, 2, \ldots, n\}$.*

4.2 Faster Method

To answer the equivalence query in $O(1)$ time, we can store the partial sums in a fully indexable dictionary [20] with the improved redundancy of [11] that supports the predecessor query in constant time albeit using $O(\sqrt{n}^{1+\epsilon})$ bits of space. However, we show that the predecessor can be supported in constant time using only $O(\sqrt{n})$ bits. As in the previous section, we store the two sequences:

- $\vec{\delta}$ that consists of $\delta_1 = s_1 n_1$ and $\delta_i = s_i n_i - s_{i-1} n_{i-1}$, for $i = 2$ to k,
- $\vec{n}$ that consists of n_i, for $i = 1$ to k,

and their shadow sequences. In addition, we store an array A, where $A[i] = \max\{j \mid \sum_{t=1}^{j} s_t n_t \leq i(i+1)/2\}$, for $i = 1$ to $\sqrt{2n}$. Now we claim:

Lemma 11. *The predecessor $p(x)$ of an integer x in the sequence $\sum_{t=1}^{i} s_t n_t$, $i = 1$ to k is in the range $[A[\lfloor\sqrt{2x}\rfloor - 1], A[\lfloor\sqrt{2x}\rfloor - 1] + 5]$.*

Proof. Let $i = \lfloor\sqrt{2x}\rfloor - 1$, notice that

$$i(i+1)/2 \leq (\sqrt{2x} - 1)\sqrt{2x}/2 \leq x$$

and

$$x \leq \sqrt{2x}(\sqrt{2x} + 1)/2 \leq (i+2)(i+3)/2$$

For $j = A[i] + 1$, $\sum_{t=1}^{j-1} s_t n_t \leq i(i+1)/2$, so $j - 1 \leq i$ and $j \leq i + 1$. Since $\sum_{t=1}^{j} s_t n_t > i(i+1)/2$, $s_j n_j \geq i(i+1)/(2j) \geq i/2$. Hence, $\sum_{t=1}^{j+5} s_t n_t \geq (i+2)(i+3)/2 \geq x$. $\square$

The actual value of $p(x)$ can be obtained by checking at most six numbers. Moreover, A can be stored using $O(\sqrt{n})$ bits by storing a bit vector that contains the values $A[0]$ and $(A[i] - A[i-1])$ for all $i = 1, \ldots, \sqrt{2n}$ represented in unary with a 0 separator between each two consecutive values. To get the value of $A[i]$, count the number of 1s before the i^{th} 0 in the bit vector.

In the standard word RAM model, computing $\sqrt{x}$ is not a constant time operation. The standard Newton's iterative method uses $O(\lg\lg n)$ operations. We describe a space efficient method that uses a look-up to precomputed tables and finds $\sqrt{x}$ in constant time. We use two tables, one when the number of bits up to the most significant bit of x is odd, denoted by O, and one when the number of bits is even, denoted by E. For $i = 1, \ldots, \lceil\sqrt{2n}\rceil$, we store in $E[i]$ the value of $\lfloor\sqrt{i2^{\lceil\lg i\rceil}}\rfloor$, and in $O[i]$ the value of $\lfloor\sqrt{i2^{\lceil\lg i\rceil - 1}}\rfloor$.

Lemma 12. *Let $i = a_i 2^{\lceil\frac{\lceil\lg i\rceil}{2}\rceil} + b_i$, $\lfloor\sqrt{i}\rfloor = E[a_i]$ or $E[a_i] + 1$ if $\lceil\lg i\rceil$ is even and $O[a_i]$ or $O[a_i] + 1$ otherwise.*

Proof. Notice that $E[a_i + 1] \le E[a_i] + 1$ and $O[a_i + 1] \le O[a_i] + 1$. Moreover, since $a_i 2^{\lceil \frac{\lceil \lg i \rceil}{2} \rceil} \le x \le (a_i + 1)2^{\lceil \frac{\lceil \lg i \rceil}{2} \rceil}$our claim holds. $\square$

The value of $\lfloor \sqrt{i} \rfloor$ can be computed by squaring the two values and comparing them with i. Notice that for $i \le n$, $a_i \le \lceil \sqrt{2n} \rceil$, and it can be computed by first finding the most significant bit r of i, then masking the lower half of the bits, and finally shifting the bits to the right by $\lfloor r/2 \rfloor$. Since the most significant bit can be found in constant time with the standard word RAM model [9] and since E and O can be stored in $O(\sqrt{n})$ bits by storing them in a similar method as we stored A, we have:

Lemma 13. ([16]) *For $i \le n$, $\lfloor \sqrt{i} \rfloor$ can be computed in constant time using a precomputed table of $O(\sqrt{n})$ bits.*

For each i, where at least one of δ_i's bits locations in $\vec{\delta}$ is a multiple of $(\epsilon \lg n)$, store the partial sum value $\sum_{j=1}^{i} (s_j n_j)$ and the value of $s_i n_i$. Moreover, for every possible sequence of δ values $\delta_1, \delta_2, \ldots, \delta_i$ of length $(\epsilon \lg n)$ and its corresponding shadow sequence, store in a table T the values i and $\sum_{j=1}^{i} (\sum_{k=1}^{j} \delta_k)$. To compute $\sum_{j=1}^{i} (s_j n_j)$ for an arbitrary index i, find the biggest index $k \le i$ that has it's partial sum value stored. Notice that $\sum_{j=1}^{i} (s_j n_j) = \sum_{j=1}^{k} (s_j n_j) + (i - k) s_k n_k + \sum_{j=k+1}^{i} (\sum_{l=k+1}^{i} \delta_l)$. Since these values can be obtained using table lookup on T, we can compute the partial sum at an arbitrary index in constant time. Moreover, we can compute the value of $s_i n_i$ for an arbitrary index i by computing the partial sum at $i - 1$ and subtracting it from the partial sum at i. Finally, we can compute s_i by computing $s_i n_i$ and dividing it by n_i. By choosing $\epsilon < 1/4$, the size of T becomes $o(\sqrt{n})$ bits, and we obtain the following theorem:

Theorem 14. ([16]) *Given a partition of an n-element set into equivalence classes $O(\sqrt{n})$ bits are necessary and sufficient for storing the partition and to answer the equivalence query in constant time if each element is to be given a unique label in the range $\{1, 2, \ldots, n\}$.*

Chapter 5

Data Structures with Extended Label Space

In this chapter, we study the trade-off between label space and auxiliary data structure size for the problem of supporting equivalence queries.

5.1 Data Structures with Label Space cn

In this section, we move on to designing data structures where the n-element s can be freely labelled with unique labels in the range of 1 to cn. Our work is motivated by the fact that unless n is a power of two or a little less than a power of two, we can increase the address space by a constant factor without increasing the number of bits required to store each address. The queries can be answered by looking at an auxiliary data structure. We are interested in time and space efficient data structures that are within a constant factor from the information theoretic lower bound. We first assign an implicit ordering of the elements. Each element gets a label according to this ordering, and the queries are answered by looking at these labels and the auxiliary data structure. Theorems 15 and 16 are to lead the reader to our main result in this section, which is Theorem 17.

Theorem 15. *Given a partition of an n-element set into equivalence classes, $O(\lg^2 n)$ bits are sufficient for storing the partition and to answer the equivalence query in $O(1)$ time if the elements are to be given unique labels in the range $\{1, 2, \ldots, cn\}$ for any constant $c > 1$.*

Proof. Given a partition, first round the size of each class to the nearest power of c. The required address space will increase from n to at most cn because the size of each class increased by at most a factor of c. Let k be the number of distinct class sizes (note that $k \leq \log_c n$). For $i = 1$ to k, let s_i be the distinct sizes of the classes and let n_i be the number of classes of size s_i. Order the classes in non-decreasing order by s_i so that for $i = 1$ to $k-1$, $s_i \leq s_{i+1}$. Assign labels to elements such that classes of size s_i are assigned values from $\sum_{j=1}^{i-1} s_j n_j + 1$ to $\sum_{j=1}^{i} s_j n_j$. Our data structure consists of:

- a fusion tree [9] T storing the values $\sum_{j=1}^{i} s_j n_j$ for $i = 1$ to k. Since $k \leq \log_c n$, T supports predecessor queries in $O(1)$ time.
- a sequence that consists of n_i, for $i = 1$ to k.
- a sequence that consists of s_i, for $i = 1$ to k.

Given an element labeled x, we can find the predecessor $p(x)$ of x in $O(1)$ time by querying T. Given x and y, if $p(x) \neq p(y)$ then x and y belong to different equivalence classes. If $p(x) = p(y)$, then we know that x and y belong to equivalence classes of the same size. They are in the same equivalence class if and only if $\lceil (x - \sum_{i=1}^{p(x)} s_i n_i)/n_{p(x)+1} \rceil$ is equal to $\lceil (y - \sum_{i=1}^{p(y)} s_i n_i)/n_{p(y)+1} \rceil$. □

To further reduce the size of the data structure, round the size of each class to the nearest power of $\sqrt{c}$, then round each n_i value to the nearest power of $\sqrt{c}$. These operations will increase the address space by at most a factor of $(\sqrt{c})^2 = c$. To store the equivalence relation, it is sufficient to store the logarithm of the n_i values, which can be done in $O(\lg n \lg \lg n)$ bits.

Theorem 16. *Given a partition of an n-element set into equivalence classes, $O(\lg n \lg \lg n)$ bits are sufficient for storing the partition and to answer the equivalence queries in constant time if each element has to be given a unique label in the range $\{1, 2, \ldots, cn\}$ for any constant $c > 1$.*

Next, we describe how to obtain a data structure whose size is $O(\lg n)$ bits. First we round the size of each class to the nearest power of $l = 2^{1/d}$, where d is the smallest integer such that $l \leq \sqrt{c}$, increasing the label space by at most a factor of $\sqrt{c}$. Next, we make sure that the distinct class sizes fill non-intersecting parts whose size is a multiple of $s = \lfloor kn/(\lceil \log_l(n) \rceil) \rfloor$ where $k = c - \sqrt{c}$. Since we have at most $\lceil \log_l(n) \rceil$ distinct class sizes, this operation will increase our address space by at most kn, so the new address space will have an upper bound of

$$\begin{aligned} n(\sqrt{c}+k) &= n(\sqrt{c}+(c-\sqrt{c})) \\ &= cn \end{aligned}$$

as desired. Let the size of the part filled by $\lfloor l^i \rfloor$ be equal to $c_i s$ (note that c_i can be equal to 0). Notice that

$$\begin{aligned} \sum_{i=0}^{\lceil \log_l(n) \rceil} c_i &\le cn/s \\ &= c\lceil \log_l(n) \rceil / k \\ &\in O(\lg(n)). \end{aligned}$$

To represent the equivalence class relation it is sufficient to store the c_i values. Our data structure consists of a single bit vector ψ that stores the c_i values in unary with a 0 separator between each two consecutive values. We also store a select structure on ψ to identify the 1s and 0s quickly, and we store a rank structure to count the 1s and 0s quickly. Finally, we store the values of $2^{i/d}$ for $i = \{1, \ldots, d-1\}$. The space required is $O((c/k+1)\lceil \log_l(n) \rceil) \in O(\lg n)$ bits.

Assign labels to elements such that classes of size $\lfloor l^i \rfloor$ are assigned values from the range $\sum_{j=0}^{i-1} c_j s$ to $(\sum_{j=0}^{i} c_j s) - 1$.

Implementing the equivalence query Now given an element labeled x, we can determine the size of the equivalence class that x belongs to by getting the number of zeroes i before the $\lfloor x/s \rfloor$-th 1 in ψ. Once we find i we know that x belongs to an equivalence class of size $\lfloor l^i \rfloor$. Given two elements x and y, if they belong to classes with different sizes, then x and y are not in the same equivalence class.

If x and y both belong to a class of size $\lfloor l^i \rfloor$, then calculate j the number of 1s before the i-th 0 in ψ. x and y are in the same equivalence class if and only if $\lfloor (x-js)/\lfloor l^i \rfloor \rfloor = \lfloor (y-js)/\lfloor l^i \rfloor \rfloor$. To calculate $\lfloor l^i \rfloor$, notice that $\lfloor l^i \rfloor = \lfloor 2^{i/d} \rfloor = \lfloor 2^{\lfloor i/d \rfloor} 2^{(i \bmod d)/d} \rfloor$. We can calculate $2^{\lfloor i/d \rfloor}$ in constant time using bitwise left shift operator ($2^{\lfloor i/d \rfloor} = 1 << \lfloor i/d \rfloor$), and we have the value of $2^{(i \bmod d)/d}$ stored. Thus, we can calculate $\lfloor l^i \rfloor$ in constant time. All other operations take constant time, so we obtain the following theorem:

Theorem 17. *Given a partition of an n-element set into equivalence classes, $\Theta(\lg n)$ bits are sufficient for storing the partition and to answer the equivalence query in $\Theta(1)$ time if each element is to be given a unique label in the range $\{1, 2, \ldots, cn\}$ for any constant $c > 1$.*

5.2 Data Structures with Label Space $f(n) \cdot n$

In this section, we generalize the techniques presented in the previous section to design a data structure where the n-element s can be freely labelled with unique labels in the range of 1 to $f(n)n$ where $f(n)$ is an increasing function such that $f(n) \in \omega(1)$ and $f(n) \in O(\lg(n))$.

Theorem 18. *Given a partition of an n-element set into equivalence classes and a function $f(n)$ where $f(n) \in \omega(1)$ and $f(n) \in O(\lg(n))$, $O(\lg(n)/\lg(f(n)))$ bits are sufficient for storing the partition and to answer the equivalence query in $O(1)$ time if the elements are to be given unique labels in the range $\{1, 2, \dots, f(n)n\}$.*

Proof. Let $l = 2^{\lfloor \lg(f(n)/2) \rfloor}$, we round the size of each class to the nearest power of l, increasing the label space by at most a factor of l. Then, we make sure that the distinct class sizes fill non-intersecting parts whose size is a multiple of $s = \lfloor 2^{\lceil \lg(n) \rceil} f(n)/(4\lceil \log_l(n) \rceil) \rfloor$. Since we have at most $\lceil \log_l(n) \rceil$ distinct class sizes, this operation will increase our address space by at most $2^{\lceil \lg(n) \rceil} f(n)/4$, so the new address space will have an upper bound of:

$$\begin{aligned} n(f(n)/2) + 2^{\lceil \lg(n) \rceil} f(n)/4 &\le n(f(n)/2) + 2nf(n)/4 \\ &= f(n)n \end{aligned}$$

as we desired. Let the size of the part filled by l^i be equal to $c_i s$. Notice that

$$\begin{aligned} \sum_{i=0}^{\lceil \log_l(n) \rceil} c_i &\le f(n)n/s \\ &\le 4\lceil \log_l(n) \rceil \\ &\in (\lg(n)/\lg(f(n))) \end{aligned}$$

To represent the equivalence class relation, we store the value of $\lceil \lg(n) \rceil$ and $f(n)$ using $O(\lg \lg(n))$ bits. Moreover, we store the unary representation of the c_i values with a 0 separator between each two consecutive values in a single bit vector ψ. We also store a select structure on ψ to identify the 1s and 0s quickly, and we store a rank structure to count the 1s and 0s quickly. Assign labels to elements such that classes of size l^i are assigned values from the range $\sum_{j=0}^{i-1} c_j s$ to $(\sum_{j=0}^{i} c_j s) - 1$. To answer the equivalence query, we first calculate the value of s and l. Now given an element labeled x, we can determine the size of the equivalence class that x belongs to by getting the number of zeroes i before the $\lfloor x/s \rfloor$-st 1 in ψ. Once we find i we know that x belongs to an equivalence class of size l^i (we can compute l^i in constant time since l is a power of 2). Given two elements

x and y, if they belong to classes with different sizes, then x and y are not in the same equivalence class. If x and y both belong to a class of size l^i, then calculate j the number of 1s before the i-th 0 in ψ. Then x and y are in the same equivalence class if and only if $\lfloor (x - js)/l^i \rfloor = \lfloor (y - js)/l^i \rfloor$.

$\square$

Chapter 6

Lower Bounds

In this chapter, we show that the space bound of Theorem 17 and Theorem **??** is optimal for the range of label space used. Without loss of generality, to make our calculations easier, we assume that n is a power of 2.

For any constant $c > 1$, let S_{cn} be the set of all partitions of $\lfloor cn \rfloor$ and S_n the set of all partitions of n. While one partition of $\lfloor cn \rfloor$ can dominate many partitions of n, we argue first that at least $(\lg n)/2c$ partitions of $\lfloor cn \rfloor$ are necessary to dominate all partitions of n. Let $\mathcal{S}$ be the smallest set of partitions of $\lfloor cn \rfloor$ that dominates all the partitions of n. Our first claim is that:

Lemma 19. $|\mathcal{S}| \geq \lg n/(2c)$.

Proof. Consider the subset Q of S_n defined as follows:

- Q contains $\lg n + 1$ partitions, and
- the i^{th} partition, for $i = 0$ to $\lg n$, q_i of Q contains $n/2^i$ classes of size 2^i.

Let p be a partition of $\lfloor cn \rfloor$ that dominates partitions $q_{j_1}, q_{j_2}, \ldots, q_{j_m}$ of Q where $j_1 < j_2 < \ldots < j_m$.

To dominate q_{j_m}, p must contain at least $n/2^{j_m}$ classes of size 2^{j_m}. Since p dominates $q_{j_{i+1}}$, for any $1 \leq i < m$, there must exist at least $n/2^{j_{i+1}}$ classes of size greater then 2^{j_i} in p. Therefore, for p to dominate q_{j_i}, p must contain at least

$$\begin{aligned} n/2^{j_i} - n/2^{j_{i+1}} &\geq n/2^{j_i} - n/2^{j_i+1} \\ &\geq n/2^{j_i+1} \end{aligned}$$

additional classes of size greater than or equal to 2^{j_i}. Consequently:

$$\begin{aligned} cn &\geq n + \sum_{k=1}^{m-1} 2^{j_k} n / 2^{j_k+1} \\ &\geq \sum_{k=1}^{m} n/2 \\ &\geq mn/2 \end{aligned}$$

and $m \leq 2c$. Thus any partition of $\lfloor cn \rfloor$ can dominate at most $2c$ partitions of Q, and to dominate Q we need a minimum of $(\lg n + 1)/(2c) \in \Omega(\lg n)$ partitions of $\lfloor cn \rfloor$. Since Q is a subset of S_n, our claim holds. □

Extending the above argument, we show

Lemma 20. *Let $k \geq 1$ be any integer such that $\lg(n/k) > 2ck$, then $|\mathcal{S}| \geq \binom{\lfloor \lg(n/k) \rfloor}{k} / \binom{\lfloor 2ck \rfloor}{k}$.*

Proof. Without loss of generality and to make our calculations easier, we assume that k is a power of 2. Divide n into k parts each of size n/k. Let Q be the set formed by filling each part with a distinct power of 2, clearly $|Q| = \binom{\lfloor \lg(n/k) \rfloor}{k}$.

Let p be a partition of $\lfloor cn \rfloor$ such that p dominates m parts filled by the following distinct powers of 2: $2^{j_1}, 2^{j_2}, \ldots, 2^{j_m}$ where $j_1 < j_2 < \ldots < j_m$.

To dominate the part filled by 2^{j_m}, p must contain at least $n/(k2^{j_m})$ classes of size 2^{j_m}. Since p dominates the part filled by $2^{j_{i+1}}$, for any $1 \leq i < m$, there must exist at least $n/(k2^{j_{i+1}})$ classes of size greater than 2^{j_i} in p. Therefore, for p to dominate the part filled by 2^{j_i}, p must contain at least

$$\begin{aligned} n/(k2^{j_i}) - n/(k2^{j_{i+1}}) &\geq n/(k2^{j_i}) - n/(k2^{j_i+1}) \\ &\geq n/(k2^{j_i+1}) \end{aligned}$$

additional classes of size greater than or equal to 2^{j_i}. Consequently

$$\begin{aligned} cn &\geq n/k + \sum_{i=1}^{m-1} 2^{j_i} n/(k2^{j_i+1}) \\ &\geq \sum_{i=1}^{m} n/(2k) \\ &\geq mn/(2k) \end{aligned}$$

and $m \leq 2ck$. Thus any partition of $\lfloor cn \rfloor$ can dominate at most $\lfloor 2ck \rfloor$ distinct parts, and any partition of $\lfloor cn \rfloor$ can dominate at most $\binom{\lfloor 2ck \rfloor}{k}$ partitions of Q. Hence, to dominate Q we need a minimum of $\binom{\lg(\lfloor n/k)\rfloor}{k}/\binom{\lfloor 2ck \rfloor}{k}$ partitions of $\lfloor cn \rfloor$. Since Q is a subset of S_n our claim holds. $\square$

The information theoretic lower bound for space to represent the equivalence class information is given by $\lg(|\mathcal{S}|) \geq \lg(\binom{\lfloor \lg(n/k) \rfloor}{k}/\binom{\lfloor 2ck \rfloor}{k})$, if we choose $k = \lceil \lg n/4c \rceil$ we get our desired bound $\lg(|\mathcal{S}|) \in \Omega(\lg n)$.

Theorem 21. *Given a partition of an n-element set into equivalence classes, $\Theta(\lg n)$ bits are necessary and sufficient for storing the partition if each element is to be given a unique label in the range $\{1, 2, \ldots, cn\}$ for any constant $c > 1$. Moreover, the equivalence query in such a structure can be answered in $O(1)$ time.*

Chapter 7

Conclusion and Future Work

In this thesis, we first reviewed the succinct representation of static bit vectors and we showed how to support the operations *rank*, *select*, *successor* and *predecessor* in constant time using only $o(n)$ additional bits. Then we covered the *Direct Equivalence Query Problems*. We showed that a label space of $\lg n + \lg \lg n - \Theta(1)$ bits is necessary and sufficient for the *Direct Equivalence Query Problem*, and a label space of $2 \lg n - 1$ bits is necessary and sufficient for the *Online Direct Equivalence Query Problem*.

Then we reviewed data structures for representing equivalence class information once the label space is n. We showed that a data structure of size $\Theta(\sqrt{n})$ bits is necessary and sufficient for storing the equivalence class information once the label space is n, and we showed that the equivalence query can be supported in $\Theta(1)$ time using such structure.

Finally, we discussed the trade-off between label space and auxiliary space for the fundamental problem of supporting equivalence queries. Our main result is to show that once labels are assigned from the range cn a data structure whose size is $\Theta(\lg n)$ bits is necessary and sufficient to represent the equivalence classes. Our scheme allows an implicit labeling of elements and supports equivalence queries in $\Theta(1)$ time. The main motivation behind our work is that when n is not a power of 2 or slightly less than a power of 2, we can increase the address space without increasing the number of bits required to store each label. Thus, for most values of n, our result is achieved using an optimal number of bits, which is $\lceil \lg n \rceil$ bits.

Apart from providing, what we believe, a non-trivial data structure requiring only $\Theta(\lg n)$ bits, we have also touched upon the interesting tradeoff issue between auxiliary space and the label space. As there is a huge body of research in ‘labeling schemes’ (see [2]), investigation into such a tradeoff for other labeling schemes maybe interesting.

References

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. The design and analysis of computer algorithms, 1974. *Reading: Addison-Wesley*, pages 207–209, 1987.

[2] Stephen Alstrup, Philip Bille, and Theis Rauhe. Labeling schemes for small distances in trees. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 689–698, 2003.

[3] Jérémy Barbay, Luca Castelli Aleardi, Meng He, and J. Ian Munro. Succinct representation of labeled graphs. In *Algorithms and Computation*, pages 316–328. Springer, 2007.

[4] David Benoit, Erik D. Demaine, J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005.

[5] Andrej Brodnik and J. Ian Munro. Membership in constant time and almost-minimum space. *SIAM Journal on Computing*, 28(5):1627–1640, 1999.

[6] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Noy Rotbart. Dynamic and multi-functional labeling schemes. *arXiv preprint arXiv:1404.4982*, 2014.

[7] Arash Farzan and J. Ian Munro. Succinct representations of arbitrary graphs. In *Algorithms-ESA 2008*, pages 393–404. Springer, 2008.

[8] Arash Farzan and J. Ian Munro. A uniform paradigm to succinctly encode various families of trees. *Algorithmica*, 68(1):16–40, 2014.

[9] Michael L Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of computer and system sciences*, 47(3):424–436, 1993.

[10] Richard F. Geary, Rajeev Raman, and Venkatesh Raman. Succinct ordinal trees with level-ancestor queries. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 1–10, 2004.

[11] Roberto Grossi, Alessio Orlandi, Rajeev Raman, and S. Srinivasa Rao. More haste, less waste: Lowering the redundancy in fully indexable dictionaries. In *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, pages 517–528, 2009.

[12] Godfrey H Hardy and Srinivasa Ramanujan. Asymptotic formulae in combinatory analysis. *Proceedings of the London Mathematical Society*, 2(1):75–115, 1918.

[13] Guy Joseph Jacobson. *Succinct static data structures.* PhD thesis, Carnegie Mellon University, 1988.

[14] Sampath Kannan, Moni Naor, and Steven Rudich. Implicat representation of graphs. *SIAM Journal on Discrete Mathematics*, 5(4):596–603, 1992.

[15] Michal Katz, Nir A Katz, Amos Korman, and David Peleg. Labeling schemes for flow and connectivity. *SIAM Journal on Computing*, 34(1):23–40, 2004.

[16] Moshe Lewenstein, J. Ian Munro, and Venkatesh Raman. Succinct data structures for representing equivalence classes. In *Algorithms and Computation*, pages 502–512. Springer, 2013.

[17] J. Ian Munro. Tables. In *Foundations of Software Technology and Theoretical Computer Science*, pages 37–42. Springer, 1996.

[18] J. Ian Munro and Patrick K. Nicholson. Succinct posets. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 743–754, 2012.

[19] J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 118–126, 1997.

[20] Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms (TALG)*, 3(4):43, 2007.

[21] Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Inf. Process. Lett.*, 17(2):81–84, 1983.