Information Processing and Management 50 (2014) 272–283

Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/infoproman

On the compression of search trees ☆,☆☆

Francisco Claude [a], Patrick K. Nicholson [b], Diego Seco [c,*]

[a] Escuela de Informática y Telecomunicaciones, Universidad Diego Portales, Chile
[b] David R. Cheriton School of Computer Science, University of Waterloo, Canada
[c] Departamento de Ingeniería Informática y Ciencias de la Computación, Universidad de Concepción, Chile

ARTICLE INFO

Article history:
Received 7 July 2012
Received in revised form 26 August 2013
Accepted 21 November 2013
Available online 8 December 2013

Keywords:
Data structure
Compression
Integers
Random access
Search

ABSTRACT

Let $X = x_1, x_2, \ldots, x_n$ be a sequence of non-decreasing integer values. Storing a compressed representation of X that supports *access* and *search* is a problem that occurs in many domains. The most common solution to this problem uses a linear list and encodes the differences between consecutive values with encodings that favor small numbers. This solution includes additional information (i.e. samples) to support efficient searching on the encoded values. We introduce a completely different alternative that achieves compression by encoding the differences in a search tree. Our proposal has many applications, such as the representation of posting lists, geographic data, sparse bitmaps, and compressed suffix arrays, to name just a few. The structure is practical and we provide an experimental evaluation to show that it is competitive with the existing techniques.

1. Introduction

The storage of ordered sets of integers is a fundamental problem in computer science that has applications in many domains. When space is not an issue these sets can be stored in arrays, which support random access and efficient searches. However, space is a constraining factor in most domains, and the compression of these sets can save a considerable amount of space. As compression techniques are usually based on variable length encoding, random access and efficient searches become challenging.

Common solutions to this problem rely on the fact that differences between consecutive values in a sequence are often small numbers, and encode these differences with encodings that favor small numbers. In order to efficiently support random access and searches, these schemes are forced to store sampled absolute values. These samples can be thought as an index built on top of the data that requires more space the higher the sampling rate. As the sampling rate is a parameter that provides a space–time trade-off, these solutions are called *parametric*.

In this paper we propose a (*non-parametric*) compressed representation for non-decreasing sets of integers that does not require us to build an index on top of the data, and still offers efficient support for random access and searches. In this sense our method is similar to a recent proposal by Teuhola (2011), but the techniques are different and of independent interest. Let $X = x_1, x_2, \ldots, x_n$ be a sequence of non-decreasing integer values, we propose a compressed representation of X that, in logarithmic time, supports:

☆ A preliminary partial version on this work appeared in Proc. DCC'12, pp. 357–366.

☆☆ This work was supported in part by the Google US/Canada PhD Fellowship Program, the David R. Cheriton Scholarships Program, and CONICYT Fondecyt Iniciación 11130104 (first author); an NSERC of Canada PGS-D Scholarship (second author); and CONICYT Fondecyt Iniciación 11130377 and Ministerio de Ciencia e Innovación (PGE/FEDER) grant TIN2009-14560-C03-02 (third author). Work partially done while the authors were at the David R. Cheriton School of Computer Science, University of Waterloo, Canada.

* Corresponding author. Tel.: +56 (41) 220 4692; fax: +56 (41) 221 770.
E-mail addresses: fclaude@recoded.cl (F. Claude), p3nichol@cs.uwaterloo.ca (P.K. Nicholson), dseco@udec.cl (D. Seco).

0306-4573/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.ipm.2013.11.002

- access(X, i): retrieve the value at position i in X.
- search(X, t): retrieve the position of the left-most stored value greater or equal than t.[1]

This structure has applications in the representation of posting lists, which are one of the two main components of the ubiquitous inverted index (Baeza-Yates & Ribeiro-Neto, 1999; Witten, Moffat, & Bell, 1999). A basic primitive operation for these indexes is to find the intersection of posting lists, which provides a solution to handle multi-word queries. Although this primitive operation can be supported through sequential merging of the lists, some of the most efficient approaches to solve intersection queries (Hwang & Lin, 1972; Demaine, López-Ortiz, & Munro, 2000; Barbay & Kenyon, 2002; Baeza-Yates, 2004) take advantage of the fact that some lists are much shorter than the others, and make use of random access and search capabilities.

As another application, the partial sums problem can be thought of as a particular instance of this problem. In the partial sums problem we are given a sequence of n non-negative values, $Y = y_1, y_2, \ldots, y_n$, and we have to support the following operations:

- sum(Y, i): retrieve the sum of all the values up to position i.
- search(Y, t): retrieve the smallest i such that sum$(Y, i) \geqslant t$.

Note that $X = x_1, x_2, \ldots, x_n$ can be defined as a sequence of partial sums of values in the sequence $Y = y_1, y_2, \ldots, y_n$, so that $x_i = \sum_{j=1}^{i} y_j$. In this way, sum(Y, i) reduces to access(X, i) and search(Y, t) reduces to search(X, t). Partial sums can be applied, for example, to represent *Rank*/*Select* dictionaries (Okanohara & Sadakane, 2007).

The article is organized as follows. In Section 2 we survey the main solutions to this problem and describe their basic properties. In Section 3 we describe our new solution to this problem. Section 4 discusses some applications, and presents an experimental evaluation of our solution both in general and for each particular application. Finally, Section 5 concludes the paper with some brief remarks and avenues for future research.

2. Related work

As we mentioned above, common solutions to this problem encode the differences between consecutive values with encodings that favor small numbers, such as γ-codes, δ-codes, or Rice codes (see Witten et al. (1999) for a comprehensive survey). In order to efficiently support random access and searches, these schemes store sampled absolute values, and the sampling rate provides a space–time trade-off. A great deal of research has been carried out on the development of new algorithms to encode small numbers (Moffat & Stuiver, 2000; Anh & Moffat, 2005; Yan, Ding, & Suel, 2009) and also on proposing storage schemes for those samplings (Culpepper & Moffat, 2010). All these techniques can be classified as *parametric* solutions, whereas our method is *non-parametric*.

Recently, (Teuhola, 2011) presented a *non-parametric* scheme based on the encoding of a binary search tree of the sequence using a variation of the interpolative coding (Moffat & Stuiver, 1996; Moffat & Stuiver, 2000). The author turns interpolative coding into an index by an address calculation that guarantees enough space in the worst case (experiments show that the redundancy added by this address calculation is only about 1 bit per symbol). Although our proposal is also based on encoding the differences between values using a search tree, there are some important differences. First, our work is not dependent on a particular encoding (any encoding supporting efficient random access may be plugged into our structure). Second, we extend the representation, and add new interesting operations; for example, a batched searching operation used for intersecting inverted lists. We also provide a more elegant solution for representing the tree when n is not a power of 2. Furthermore, we extend this generalized result to trees of arbitrary constant degree, which is suitable for secondary memory. Finally, we describe an approach for handling general trees, with the potential of becoming dynamic.

Among all the encoding techniques for integers there is one of special interest for us because we use it as part of our solution. This technique was introduced by Brisaboa, Ladra, and Navarro (2013) and is known as Directly Addressable Codes (DACs). It performs a reorganization of variable-length codes in order to allow efficient random access. This technique solves the problem of supporting random access, but not the search problem. In order to support searches, the same sampling scheme mentioned above can be used. The main advantage of the use of DACs in these schemes is that, as they allow direct access to any position, no pointers from each sample to the subsequent code are necessary. Although this might mean a significant reduction of space, in many applications the lower compression rate achieved by these codes counteracts this improvement.

3. Differentially Encoded Search Tree (DEST)

The main idea behind our structure is to differentially encode the values inside a search tree built over the sequence of non-decreasing values, X. Every node stores the difference between its value and the one represented by its parent node. By

[1] Note that we can also return the value stored in that position, thus solving the membership variant of the problem.