Theoretical Computer Science 638 (2016) 91–97

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Fast construction of wavelet trees ☆

J. Ian Munro [a], Yakov Nekrich [a,*], Jeffrey S. Vitter [b]

[a] *Cheriton School of Computer Science, University of Waterloo, Canada*
[b] *Department of Electrical Engineering & Computer Science, University of Kansas, United States*

ARTICLE INFO

Article history:
Received 31 March 2015
Received in revised form 10 October 2015
Accepted 8 November 2015
Available online 1 December 2015

Keywords:
Wavelet trees
Compressed data structures
Compressed sequences

ABSTRACT

In this paper we describe a fast algorithm that creates a wavelet tree for a sequence of symbols. We show that a wavelet tree can be constructed in $O(n\lceil \log\sigma/\sqrt{\log n}\rceil)$ time where n is the number of symbols and σ is the alphabet size.

1. Introduction

Wavelet tree, introduced in [1], is one of the most extensively studied succinct data structures. Wavelet trees are frequently chosen as a space-efficient data structure that supports access, rank and select queries on a sequence of symbols. An access query access(i, X) returns the i-th symbol in a sequence X; a rank query $\text{rank}_a(i, X)$ computes how many times a symbol a occurs in the prefix $X[1..i]$ of X; select query $\text{select}_a(i, X)$ finds the position where a occurs for the i-th time. Since wavelet trees can efficiently support operations rank and select, they can be used in succinct representations of graphs [2], strings, points and other geometric objects on a grid, full-text indexes [1,3], data structures for document retrieval [4], XML documents [5], and binary relations [6]. It was also shown that wavelet trees and their variants can be used to answer various queries on points and other geometric objects [7]. We refer to recent extensive surveys of Navarro [8] and Makris [9] for a description of these and other applications of wavelet trees. In this paper we describe the first algorithm that constructs a wavelet tree in $o(n\log\sigma)$ time. n^0 We show how to construct a wavelet tree in $O(n\lceil \frac{\log\sigma}{\sqrt{\log n}}\rceil)$ time.

Let X be a sequence of length n over an alphabet of size σ. We can assume w.l.o.g. that the i-th element $X[i]$ of X is an integer in the range $[1, \sigma]$. Essentially constructing a wavelet tree for a sequence X requires re-grouping the bits of X into a bit sequence of total length $n\log\sigma$. Since different bits of an element $X[i]$ are stored in different parts of the bit sequence, it appears that we need $\Omega(n\log\sigma)$ time to construct a wavelet tree. In this paper we show that the cost of the straightforward solution can be reduced by an $O(\sqrt{\log n})$ factor. The main idea of our method is usage of bit parallelism, i.e. we use bit operations to keep $\Omega(1)$ elements of X in one word and perform certain operations on elements packed into one word in constant time. Suppose that we can pack L symbols of a sequence X into one machine word. Then we can generate the wavelet tree for the resulting sequence of symbols in $O(n(\log\sigma/L))$ time by processing $O(L)$ symbols in constant time.

☆ An early version of this work appeared in SPIRE 2014.
* Corresponding author.
E-mail addresses: imunro@uwaterloo.ca (J. Ian Munro), ynekrich@uwaterloo.ca (Y. Nekrich), jsv@ku.edu (J.S. Vitter).

http://dx.doi.org/10.1016/j.tcs.2015.11.011
0304-3975/

Previous and related work. Since wavelet trees were introduced in 2003 [1], a large number of papers that use this data structure appeared in the literature [3,10–16]. A more extensive list of previous results can be found in surveys of Makris [9] and [8]. In spite of a significant number of previous papers, no results for constructing a wavelet tree in $o(n\log\sigma)$ time were previously described. Algorithms that generate a wavelet tree and use little additional workspace were considered by Claude et al. [17] and Tischler [18].

Chazelle [19] described a linear space ($O(n\log n)$-bit) geometric data structure that answers certain kinds of two-dimensional range searching queries. Data organization in [19] is the same as in wavelet tree. n^{o} quite similar to the approach of wavelet trees. We remark, however, that the intended usage of the wavelet tree and Chazelle's data structure are different. The data structure of Chazelle [19] supports different kinds of geometric queries and uses $O(n\log n)$ space to store n two-dimensional points. On the other hand, the wavelet tree, as described in [1] and later works, uses $n\log\sigma$ bits to store a sequence of size n over an alphabet of size σ; the space usage can also be reduced to nH_0 bits, where H_0 is the zero-order entropy of the original sequence. Some other linear-space geometric data structures [20] also use similar ways of structuring data. By the same argument, we need $O(n\log n)$ time to construct these data structures. Chan and Pătraşcu [21] showed that bit parallelism can be used to obtain linear-space data structures with faster construction time. In [21] they describe data structures that use linear space and can be constructed in $O(n\sqrt{\log n})$ time. Their approach is based on recursively reducing the original problem to several problems of smaller size. When point coordinates are sufficiently small, we can pack L points into one machine word and process data associated to L points in constant time. Very recently, the problem of constructing a wavelet tree was addressed by Babenko et al. [22]; the result presented in [22] and published after the conference version of this paper, is equivalent to our result.

In this paper we show how bit parallelism can be applied to speed-up the construction of the standard wavelet tree data structure. Our simple two-stage approach improves the construction time of the wavelet tree by $O(\sqrt{\log n})$. After recalling the basic concepts in Section 2, we describe the main algorithm and its variants in Section 3. In Section 4 we show how we can construct secondary data structures stored in the wavelet tree nodes. Finally, in Section 5 we show how our result can be used to speed-up the construction algorithm for a geometric data structure that answers two-dimensional orthogonal range maxima queries.

2. Wavelet tree

Let X denote a sequence over alphabet $\Sigma=\{1,\dots,\sigma\}$. The standard wavelet tree for X is a balanced binary tree with bit sequences stored in each internal node. These bit sequences can be obtained as follows: we start by dividing the alphabet symbols into two subsets Σ_0 and Σ_1 of equal size, $\Sigma_0=\{1,\dots,\sigma/2\}$ and $\Sigma_1=\{\sigma/2+1,\dots,\sigma\}$. Let X_0 and X_2 denote the subsequences of X induced by symbols from Σ_0 and Σ_1 respectively. The bit sequence $X(v_R)$ stored in the root v_R of the wavelet tree indicates for each symbol $X[i]$ whether it belongs to X_0 or X_1: $X(v_R)[i]=0$ if $X[i]$ is in X_0 and $X(v_R)[i]=1$ if $X[i]$ is in X_1. The left child of v_R is the wavelet tree for X_0 and the right child of v_R is the wavelet tree for X_1.

A symbol from an alphabet Σ can be represented as a bit sequence of length $\lfloor\log\sigma\rfloor$ or $\lceil\log\sigma\rceil$. Bit sequences $X(u)$ in the nodes of the wavelet tree consist of the same bits as the symbols in X, but the bits are ordered in a different way. The sequence $X(v_R)$ contains the first bit from each symbol $X[i]$ in the same order as symbols appear in X. Let v_l and v_r be the left and the right children of v_R. The sequence $X(v_l)$ contains the second bit of every symbol in X_0. That is, $X(v_l)$ contains the second bit of every symbol $X[i]$, such that the first bit of $X[i]$ is 0. $X(v_r)$ contains the second bit of every $X[i]$ such that the first bit of $X[i]$ is 1, etc.

Some generalizations of the wavelet tree often lead to improved results. We can consider t-ary wavelet tree for $t=\log^{\varepsilon}n$ and a small constant $\varepsilon>0$. In this case the original alphabet Σ is divided into t parts $\Sigma_0, \dots, \Sigma_{t-1}$. The sequence $X(v_R)$ in the root node is a sequence over an alphabet $\{0,\dots,t-1\}$ such that $X(v_R)[i]=j$ iff $X[i]$ is a symbol from Σ_j for $1\le j\le t$. Let X_j be the subsequence of X induced by symbols from Σ_j. The j-th child v_j of v_R is the root of the wavelet tree for X_j. The advantage of the t-ary wavelet tree is that the tree height is reduced from $O(\log\sigma)$ to $O(\log\sigma/\log\log n)$. Another useful improvement is to modify the shape of the tree so that the average leaf depth is (almost) minimized. Finally we can also keep the binary or t-ary sequences $X(u)$, stored in the nodes, in compressed form. Two latter improvements enable us to store a sequence X in asymptotically optimal space.

3. Constructing a wavelet tree

In this section we describe our algorithm for constructing a wavelet tree. Our method uses bit parallelism in a way that is similar to [21]. However a recursive algorithm employed in [21] to reduce the problem size is not necessary. Our algorithm consists of two stages. During the first stage we construct an L-ary wavelet tree $\mathcal{T}^g$ for $L=2^{\sqrt{\log n}}$. That is, each internal node $u\in\mathcal{T}^g$ has L children. To avoid tedious details, we assume that L is an integer that divides σ. An L-ary wavelet tree can be defined in the same way as in Section 2. We partition the alphabet $\Sigma=\{1,\dots,\sigma\}$ into L parts $\Sigma_1, \Sigma_2, \dots, \Sigma_L$. Each Σ_i for $1\le i\le L-1$ contains σ/L alphabet symbols; the last part Σ_L contains at most σ/L symbols. The root node u_R of $\mathcal{T}^g$ contains a sequence $X^g(u_R)$. Every element of $X^g(u_R)$ is a positive integer that does not exceed L. $X^g(u_R)[i]=j$ if $X[i]$ is a symbol from Σ_j. The child u_i of u is the root node of the wavelet tree for the subsequence X_i, where X_i is the subsequence of X induced by symbols from Σ_i. An L-ary tree can be constructed in $O(\log\sigma/L)$ time. During the second stage, we transform an L-ary tree into a binary tree. We replace each internal node u of $\mathcal{T}^g$ with a subtree $T(u)$ of height