Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 78 (2005) 249–256
The Journal of Systems and Software
www.elsevier.com/locate/jss

Worst case constant time priority queue ☆

Andrej Brodnik [a,b,c,*], Svante Carlsson [d], Michael L. Fredman [e], Johan Karlsson [a], J. Ian Munro [f]

[a] Department of Computer Science and Electrical Engineering, Luleå University of Technology, SE-97187 Luleå, Sweden
[b] Department of Theoretical Computer Science, IMFM, Institute of Mathematics, Physics, and Mechanics, S11000/1111, Jadranska 19, Ljubljana, Slovenia
[c] Faculty of Education, University of Primorska, Muzejski trg 2, SI-6000 Koper, Slovenia
[d] Blekinge Institute of Technology, SE-37179 Karlskrona, Sweden
[e] Department of Computer Science, Rutgers University, New Brunswick, NJ
[f] School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1

Received 8 August 2003; received in revised form 18 September 2004; accepted 20 September 2004

Abstract

We present a new data structure of size $3M$ bits, where M is the size of the universe at hand, for realizing a *discrete priority queue*. When this data structure is used in combination with a new memory topology it executes all discrete priority queue operations in O(1) worst case time. In doing so we demonstrate how an unconventional, but practically implementable, memory architecture can be employed to sidestep known lower bounds and achieve constant time performance.

MSC: 68Q05; 68Q25; 68Q65
Keywords: Data structure; Discrete priority queue; Split tagged tree

1. Introduction

In this paper we reexamine the well known "discrete priority queue" problem of van Emde Boas et al. (1997). Operating over the bounded universe of integers $\mathscr{M} = [0, \ldots, M-1]$, the usual operations of *Insert* and *ExtractMin* are supported, as are the additional operations of finding and removing any value, and finding *Predecessor*(e) and *Successor*(e). The last two operations determine, respectively, the largest element present that is less than e, and the smallest greater than e. The problem is referred to by Mehlhorn et al. (1998) as the union-split-find problem. Under this terminology, one thinks of $[0,\ldots,M-1]$ as being partitioned into subranges that can be further subdivided or merged, and that one can ask for the subrange containing a given value. We revert to the priority queue analogy and terminology of van Emde Boas et al. and more formally define the data type as:

Definition 1. The *discrete extended priority queue problem* is to maintain a set, $\mathscr{N}$ of size N, with elements drawn from an ordered bounded universe $\mathscr{M} = [0..M-1]$, and support the following operations:

`Insert`(e) $\mathscr{N} := \mathscr{N} \cup \{e\}$
`Delete`(e) $\mathscr{N} := \mathscr{N} \setminus \{e\}$

☆ These results appeared in preliminary form in the Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (Brodnik et al., 2001).

* Corresponding author. Address: Department of Theoretical Computer Science, IMFM, Institute of Mathematics, Physics, and Mechanics, S11000/1111, Jadranska 19, Ljubljana, Slovenia. Tel.: +386 5 626 01 02/1 476 6572; fax: +386 5 627 80 05/1 25 17 281.
E-mail addresses: andrej.brodnik@upr.si, andrej.brodnik@imfm.uni-lj.si (A. Brodnik), svante@addendi.se (S. Carlsson), fredman@cs.rutgers.edu (M.L. Fredman), johan.karlsson@csee.ltu.se (J. Karlsson), imunro@uwaterloo.ca (J.I. Munro).

0164-1212/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2004.09.002

`Member`(e) Return whether $e \in \mathcal{N}$
`Min` Find the smallest element of $\mathcal{N}$
`Max` Find the largest element of $\mathcal{N}$
`DeleteMin` Delete the smallest element of $\mathcal{N}$
`DeleteMax` Delete the largest element of $\mathcal{N}$
`Predecessor`(e) Find the largest element of $\mathcal{N}$ less than e
`Successor`(e) Find the smallest element of $\mathcal{N}$ greater than e

We will refer to the predecessor of an element e as its *left neighbor* and the successor as its *right neighbor*. When talking about the *neighbors* of e we mean the left and the right neighbor. The *static* version of the problem does not include the `Insert`, `Delete`, `DeleteMin`, and `DeleteMax` operations, instead the set may be preprocessed. We let m denote $\lg M$.

1.1. Lower bounds and some matching upper bounds

Under the pointer machine model (cf. (Schönhage, 1980)) Mehlhorn et al. (1998) proved a lower bound of $\Omega(\lg \lg M)$ for the discrete priority queue problem. The stratified tree by van Emde Boas et al. provides a matching upper bound (van Emde Boas et al., 1997). More recently, Beame and Fich (2002) gave a lower bound, for the static version, of $\Omega(\min((\lg \lg M/\lg \lg \lg M), \sqrt{\lg N/\lg \lg N}))$, when restricting the memory usage to $N^{O(1)}$, under the communication game model (cf. (Miltersen, 1994, 1979)) which also applies to the cell probe (cf. (Yao, 1981)) and RAM (cf. (van Emde Boas, 1990)) models. This lower bound implies the same lower bound for the dynamic version. They also gave a matching upper bound, in the RAM model and hence the communication game and the cell probe models, for the static version of the problem. Andersson and Thorup (2000) gave a data structure and an algorithm with $O(\sqrt{\lg N/\lg \lg N})$ worst case time for the dynamic version. For the static case, Brodnik and Munro (1996) gave a data structure with O(1) worst case time for any N but using $O(M)$ space. On the other hand, Ajtai et al. (1986) presented a solution using only $O(N)$ space when $N = O(M^{1/p})$.

1.2. Model of computation

Our goal is to sidestep these lower bounds, but to retain a practically implementable model of computation. Our model is based on the RAM model of computation which includes branching and the arithmetic operations addition and subtraction. We will also need bitwise boolean operations and multiplication (cf. MBRAM (van Emde Boas, 1990)). However, we do not want the model to be unrealistic and therefore we restrict the model to only use bounded registers. The registers we use are at least m bits wide; i.e. a given memory location can store at least m-bit values and all operations are defined for arguments with at least m bits. For fixed m, this model of computation is implemented by any standard computer today.

Operations to search for *Least (Most) Significant Bit (LSB, MSB)* in a register can be implemented to run in O(1) time in our model, using a technique called *Word-Size-Parallelism* (Brodnik, 1993) (also, see (Fredman and Willard, 1993)). Hence, we let these operations be defined in the model as well. Alternatively, the use of $O(M^{\epsilon})$ extra bits permits such queries to be answered in constant time by simple table lookup.

The final aspect, and the crucial twist, of our model of computation is the notion of a word of memory. Under the standard model, a word is a sequence of bits and each bit is in one word only. We will consider a model in which a single bit may be in several different words. The notion of a "random access machine with byte overlap", *RAMBO*, was introduced by Fredman and Saks (1989). The way the bits occur, in the part of the memory where bytes overlap, has to be specified as part of the model variant.

We consider a variant which we refer to as *Yggdrasil* (see Section 2.3). The part of the memory where bytes overlap has been developed in hardware by *Priqueue AB*, as a SDRAM memory module according to the PC100 standard (Leben et al., 1999).

2. The Split Tagged Tree

In this section we introduce an abstract data structure *Split Tagged Tree (STT)* used to solve the discrete extended priority queue problem. We first define the STT and describe its properties, and later use these properties to implement the operations from Definition 1.

2.1. The split tagged tree and its properties

In a complete binary tree that has leaves for every element in a universe $\mathcal{M}$ (cf. *trie* with leaves $0..M-1$ numbered from left to right, and leaves corresponding to elements of $\mathcal{N}$ are "tagged") we define:

Definition 2. An internal node is a *splitting node* if there is at least one tagged leaf in each of its subtrees. The splitting node v is a *left splitting node* of e if e is a leaf in the left subtree of v. The first left splitting node on the path from e to the root is the *lowest left splitting node* of e. *Right splitting nodes* and the *lowest right splitting node* of e are defined symmetrically.