Finger trees: a simple general-purpose data structure

RALF HINZE
Institut für Informatik III, Universität Bonn
Römerstraße 164, 53117 Bonn, Germany
(*e-mail:* ralf@informatik.uni-bonn.de)

ROSS PATERSON
Department of Computing, City University
London EC1V OHB, UK
(*e-mail:* ross@soi.city.ac.uk)

Abstract

We introduce 2-3 finger trees, a functional representation of persistent sequences supporting access to the ends in amortized constant time, and concatenation and splitting in time logarithmic in the size of the smaller piece. Representations achieving these bounds have appeared previously, but 2-3 finger trees are much simpler, as are the operations on them. Further, by defining the split operation in a general form, we obtain a general purpose data structure that can serve as a sequence, priority queue, search tree, priority search queue and more.

1 Introduction

Lists are the functional programmer's favourite data type. They are well supported by most if not all functional programming languages. However, one could argue that this support sometimes leads programmers to use lists when a more general sequence type would be more appropriate (Okasaki, 2000). The operations one might expect from a sequence abstraction include adding and removing elements at both ends (the deque operations), concatenation, insertion and deletion at arbitrary points, finding an element satisfying some criterion, and splitting the sequence into subsequences based on some property. Many efficient functional implementations of subsets of these operations are known, but supporting more operations efficiently is difficult. The best known general implementations are very complex, and little used.

This paper introduces functional 2-3 finger trees, a general implementation that performs well, but is much simpler than previous implementations with similar bounds. The data structure and its many variations are simple enough that we are able to give a concise yet complete executable description using the functional programming language Haskell (Peyton Jones, 2003). The paper should be accessible to anyone with a basic knowledge of Haskell and its widely used extension to