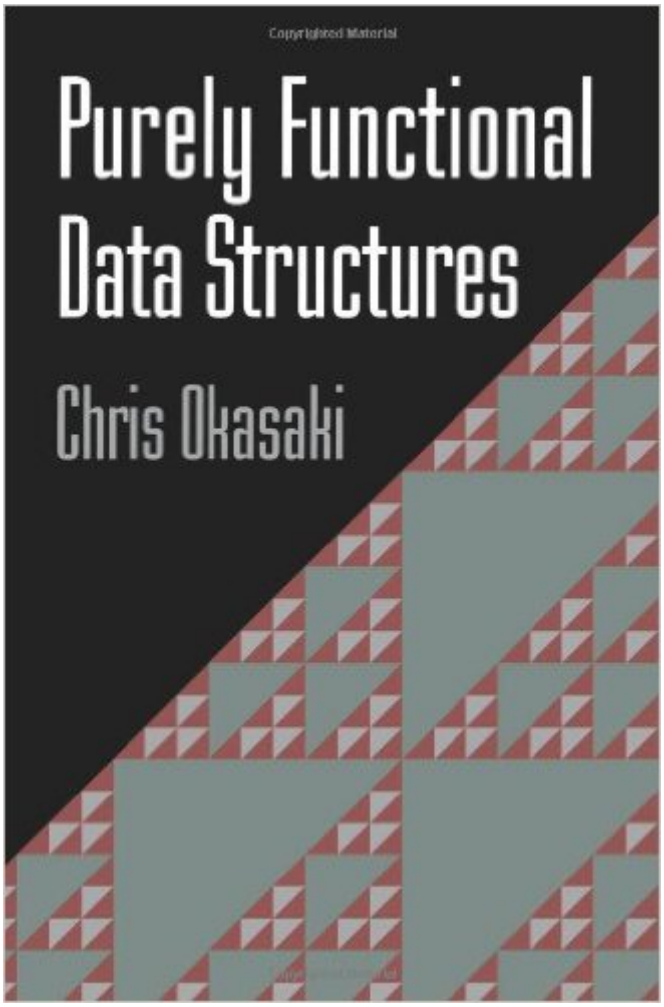
Purely Functional Data Structures



Synopsis

Most books on data structures assume an imperative language such as C or C++. However, data structures for these languages do not always translate well to functional languages such as Standard ML, Haskell, or Scheme. This book describes data structures from the point of view of functional languages, with examples, and presents design techniques that allow programmers to develop their own functional data structures. The author includes both classical data structures, such as red-black trees and binomial queues, and a host of new data structures developed exclusively for functional languages. All source code is given in Standard ML and Haskell, and most of the programs are easily adaptable to other functional languages. This handy reference for professional programmers working with functional languages can also be used as a tutorial or for self-study.

Book Information

Paperback: 232 pages
Publisher: Cambridge University Press (June 13, 1999)
Language: English
ISBN-10: 0521663504
ISBN-13: 978-0521663502
Product Dimensions: 6 x 0.5 x 9 inches
Shipping Weight: 14.6 ounces (View shipping rates and policies)
Average Customer Review: 4.5 out of 5 starsÂ Â See all reviewsÂ (20 customer reviews)
Best Sellers Rank: #272,470 in Books (See Top 100 in Books) #30 inÂ Books > Computers & Technology > Programming > Functional #35 inÂ Books > Computers & Technology > Programming > Algorithms > Data Structures #49 inÂ Books > Computers & Technology > Programming > Software Design, Testing & Engineering > Structured Design

Customer Reviews

Okasaki’s slim volume is one of the best expositions on implementing data structures & algorithms in a functional language. After taking an introductory course on functional programming, this would be the book which tells you where to go next.This book doesn’t just present a rehash/rewrite of imperative data structures, only written in a functional language. Instead, Okasaki makes sure to emphasize benefits which only functional programming can bring to the table. For example, many functional data structures can compactly represent not just their current state, but all of their past states as well--a feature called "Persistence". Also, functional newbie programmers might be

wondering why lazy vs. strict programming is a big deal, and Okasaki shows clearly where data structures can benefit from either being lazy or being strict.For the advanced reader, Okasaki also presents several powerful techniques for analyzing the runtime of algorithms, including the so-called "Banker's Method" and the "Physicist's Method" for analyzing amortized algorithms.I hope that Okasaki comes out with a 2nd edition of this book; there is one missing piece in particular which I really wish he would have included: Although he presents an EXTREMELY lucid description of how to implement Red-Black trees in a functional language, he only presented algorithms for insertion and querying. Of course, deletion from a red-black tree is the hardest part, left here, I suppose, as an exercise to the student. If you want to supply this missing piece yourself, check out a paper by Stefan Kars, "Red-black trees with types", J. Functional Programming 11(4):425-432, July, 2001.

[This review copied from my moribund blog at [...]]The typical data structures most programmers know and use require imperative programming: they fundamentally depend on replacing the values of fields with assignment statements, especially pointer fields. A particular data structure represents the state of something at that particular moment in time, and that moment only. If you want to know what the state was in the past you needed to have made a copy of the entire data structure back then, and kept it around until you needed it. (Alternatively, you could keep a log of changes made to the data structure that you could play in reverse until you get the previous state - and then play it back forwards to get back to where you are now. Both these techniques are typically used to implement undo/redo, for example.)Or you could use a persistent data structure. A persistent data structure allows you to access previous versions at any time without having to do any copying. All you needed to do at the time was to save a pointer to the data structure. If you have a persistent data structure, your undo/redo implementation is simply a stack of pointers that you push a pointer onto after you make any change to the data structure.This can be quite useful--but it is typically very hard to implement a persistent data structure in an imperative language, especially if you have to worry about memory management [1]. If you're using a functional programming language--especially a language with lazy semantics like Haskell--then all your data structures are automatically persistent, and your only problem is efficiency (and of course, in your functional languages, the language system takes care of memory management).

(Hacking Freedom and Data Driven Book 2) Big Data For Beginners: Understanding SMART Big Data, Data Mining & Data Analytics For improved Business Performance, Life Decisions & More! The Data Revolution: Big Data, Open Data, Data Infrastructures and Their Consequences Purely Primitive Dolls: How to Make Simple, Old-Fashioned Dolls Starting Out with Java: From Control Structures through Data Structures (2nd Edition) (Gaddis Series) Java Software Structures: Designing and Using Data Structures Java Software Structures: Designing and Using Data Structures (3rd Edition) Starting Out with Java: From Control Structures through Data Structures (3rd Edition) Swift: Programming, Master's Handbook: A TRUE Beginner's Guide! Problem Solving, Code, Data Science, Data Structures & Algorithms (Code like a PRO in ... mining, software, software engineering,) Java Programming Box Set: Programming, Master's Handbook & Artificial Intelligence Made Easy; Code, Data Science, Automation, problem solving, Data Structures & Algorithms (CodeWell Box Sets) Ruby Programming Box Set: Programming, Master's Handbook & Artificial Intelligence Made Easy; Code, Data Science, Automation, problem solving, Data Structures & Algorithms (CodeWell Box Sets) Data Structures and Algorithms Made Easy: Data Structure and Algorithmic Puzzles Data Structures in Java: From Abstract Data Types to the Java Collections Framework Java Programming: Master's Handbook: A TRUE Beginner's Guide! Problem Solving, Code, Data Science, Data Structures & Algorithms (Code like a PRO in 24 ... design, tech, perl, ajax, swift, python) Data Structures and Algorithms Made Easy in Java: Data Structure and Algorithmic Puzzles Ruby: Programming, Master's Handbook: A TRUE Beginner's Guide! Problem Solving, Code, Data Science, Data Structures & Algorithms (Code like a PRO in 24 ... design, tech, perl, ajax, swift, python) Functional Programming in JavaScript: How to improve your JavaScript programs using functional techniques Clinical Functional MRI: Presurgical Functional Neuroimaging (Medical Radiology)