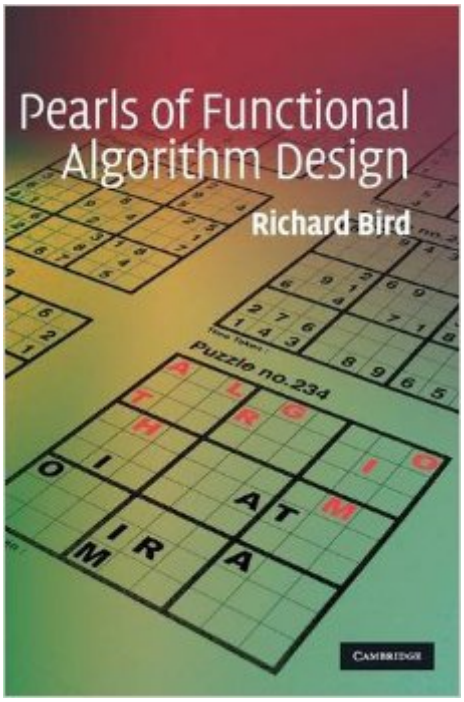
Pearls Of Functional Algorithm Design



Synopsis

Richard Bird takes a radically new approach to algorithm design, namely, design by calculation. These 30 short chapters each deal with a particular programming problem drawn from sources as diverse as games and puzzles, intriguing combinatorial tasks, and more familiar areas such as data compression and string matching. Each pearl starts with the statement of the problem expressed using the functional programming language Haskell, a powerful yet succinct language for capturing algorithmic ideas clearly and simply. The novel aspect of the book is that each solution is calculated from an initial formulation of the problem in Haskell by appealing to the laws of functional programming. Pearls of Functional Algorithm Design will appeal to the aspiring functional programmer, students and teachers interested in the principles of algorithm design, and anyone seeking to master the techniques of reasoning about programs in an equational style.

Book Information

Hardcover: 290 pages
Publisher: Cambridge University Press; 1 edition (November 1, 2010)
Language: English
ISBN-10: 0521513383
ISBN-13: 978-0521513388
Product Dimensions: 6.8 x 0.7 x 9.7 inches
Shipping Weight: 1.6 pounds (View shipping rates and policies)
Average Customer Review: 4.3 out of 5 starsÂ Â See all reviewsÂ (10 customer reviews)
Best Sellers Rank: #666,430 in Books (See Top 100 in Books) #66 inÂ Books > Computers & Technology > Programming > Functional #153 inÂ Books > Textbooks > Computer Science > Algorithms #384 inÂ Books > Computers & Technology > Programming > Algorithms

Customer Reviews

While this appears to be an excellent text, the author uses a number of mathematical symbols which are not rendered correctly in the Kindle edition. They are sometimes rendered with a question mark or a rectangular box. This can make the derivations difficult to follow. Since this book is also somewhat expensive for a Kindle edition, if you can afford it, you should probably go for the paper version (assuming the paper version is typeset correctly).

Bird has written a fine book, the functional analogue to Bentley's fine pieces. If there is one thing that's common to the functional attitude in program design, it's an emphasis on proof and logical

consistency. Bird goes into detail and carefully shows why algorithms perform, and what their costs must be in terms of time and space complexity.One quibble: the reader, to follow the arguments, will want to write the short code selections for himself, to check Bird's arguments; he'll find himself having to define a number of Unicode mathematical operators, likeU-2209, for example. But this isn't hard to do, in Haskell.Each chapter is well-written, to the point, and closely argued. In showing the beauty of Haskell in a clear way, or showing the beauty of concrete maths in a clear way, Bird has done well.

Each chapter presents a problem, a simple approach to a solution, and then shows how that simple approach can be brought all the way to implementation, using a functional programming style. There are a few things happening in this book. The author's goal to teach you to believe in the morphing properties of functional programming. In effect, he is saying: do not optimize early, express and implement your algorithm sequentially, and you will see that the implementation can be changed (refactored) to meet performance and memory usage needs. To support this mission the book presents and implements a hodgepodge of algorithms, I say hodgepodge because they do not share a common theme. Finally, the book stays within a "first order of complexity" domain: the algorithms stay focused on the problem they are solving, there are no "higher order abstractions" (e.g. monads, arrows).The lack of algorithmic or higher order theme of this book did bug me. And I would have been tempted to count this against it. Yet not long after having read much of this book, I was working with a list that I knew would no scale, hesitating to refactor it, I was happy leave it unchanged, feeling confident that I would be able to deal with it later. Part of that confidence was supported by the knowledge that that type of transformation is what this book does successfully, chapter after chapter.

I was one of those people who had learned the fundamentals of algorithms from the Programming Pearls (2nd edition) by Bentley. I have to say that since that text, many has written books about programming and algorithms in general and not one had the stuff to call it "pearl". I think this one came close to achieving the similar goal as the original one, but in the domain of functional programming. For those of you who have not read the original pearls, these books are almost like a collection of publication papers, but very well explained and throughly put into a programming environment. Another most important feature of this book (and the original pearls) is that each chapter is about a single problem that has a unique characteristic in algorithms. While most other algorithms books are organized by data structure types, these ones are focused in the category of

problems. I have to say that this is the most intuitive form for people interested in algorithms.Haskell is a perfect choice of language. However, I don't think this book is just for Haskell programmers. Some of most intriguing problems are covered in very good structure. My favorites are the coverage of saddleback search, last tail, raking suffixes and nexuses. All others topics are really well done, but these were the topics that I had hard time googling for a good explanation. Great work.

I love this book. Just like Jon Bentley's Pearls books, this rather thin book is quite literally a pearl of programming wisdom. Although it's angled to the functional paradigm (Haskell, which is a pleasure as well), it makes for provocative reading given that other languages like C++ and various scripting languages are increasingly including functional programming facilities. This belongs on every computer scientist's (and software engineer's) bookshelf.