JFP **30**, e19, 2 pages, 2020.
doi:10.1017/S0956796820000179

Book Review

Review of "Algorithms for Functional Programming" by John David Stone, Springer-Verlag, 2018

Many textbooks start out as informal notes for university courses, refined through repeated offerings. The book under review came out of the author's experiences as a lecturer at an American liberal-arts college, Grinnell, where the first course in computation used Scheme for many years.

The first chapter, "Essential Notations", rapidly introduces many features of R7RS Scheme and its libraries, augmented by some primitives written by the author. The next chapter, "The Tool Box", presents or builds a host of abstractions, such as couplers, adapters, and recursion managers (e.g., folds), which are used extensively in the rest of the text.

The ambition of this book can be seen in the topic lists for the next two chapters. "Data Structures" covers lists, sources (streams without caching, which would necessitate mutation), trees, bushes (rose trees), bags, sets, tables, and buffers (queues), all implemented mostly using pairs and lists (occasionally records). "Sorting" covers insertion sort, selection sort, quicksort, mergesort, binary search trees, treesort, red-black trees (both insertion and deletion), pairing heaps, and order statistics.

Considering that the word "Algorithms" is in the title, the book is surprisingly shy about discussing efficiency. Linear time appears briefly; the phrase "proportional to" is used a few times, and "logarithm" only once, in reference to running times. I understand the reluctance; order notation and the proper use of recurrences for analysis of running time are nettles that one hesitates to grasp. They can be avoided in a first course, but only barely, and without them, the motivation for many of these more advanced topics becomes much more difficult.

The next chapter, "Combinatorial Constructions", is possibly the most successful, perhaps because it is rich with helper functions and examples, and topics not usually covered: Cartesian products, subsequences and selections (subsets and combinations, but of an ordered list), and finally permutations and partitions, all with ranking and unranking (going from a numerical index to an element of the full answer and *vice versa*).

In the last two chapters, the author's principled insistence on purity and layers of abstraction is at odds with the material. "Graphs" presents depth-first and breadth-first traversal, spanning trees, shortest paths (Bellman-Ford, Dijkstra, and Floyd-Warshall), and maximum flow (Ford-Fulkerson). "Sublist Search" is chiefly concerned with Knuth–Morris–Pratt and Boyer–Moore string search. These are topics for which pure functional programming does not have a good story. One should be willing to sacrifice efficiency for

clarity in the study of algorithms and data structures, but at this point we are not permitted that trade-off.

Naïve graph traversal takes time proportional to the product of the number of edges and the number of vertices. Naïve string search takes time proportional to the product of the length of the pattern and the length of the text. The more complicated algorithms replace "product" with "sum" in those statements. That promise is not explicitly made here, but I don't think the implementations (which use flat lists under the hood) achieve it, and the presentations are not clearer than ones that judiciously use mutation (perhaps hiding it under a pure interface).

It is unfortunate that the publisher has chosen to present only a tiny amount of preview information on the Web. This book is sufficiently unorthodox that both the author and potential readers would be better served if the publisher made available a full sample chapter.

Conflict of interest

None.

Supplementary materials

To view supplementary material for this article, please visit https://doi.org/10.1017/S0956796820000179

PRABHAKAR RAGDE
University of Waterloo
Waterloo
Ontario N2L 3G1, Canada
E-mail: plragde@uwaterloo.ca