Finger tree

From Wikipedia, the free encyclopedia

A **finger tree** is a purely functional data structure used in efficiently implementing other functional data structures. A finger tree gives amortized constant time access to the "fingers" (leaves) of the tree, where data is stored, and also stores in each internal node the result of applying some associative operation to its descendants. This "summary" data stored in the internal nodes can be used to provide the functionality of data structures other than trees. For example, a priority queue can be implemented by labeling the internal nodes by the minimum priority of its children in the tree, or an indexed list/array can be implemented with a labeling of nodes by the count of the leaves in their children.

Finger trees can provide amortized O(1) pushing, reversing, popping, O(log n) append and split; and can be adapted to be indexed or ordered sequences. And like all functional data structures, it is inherently persistent; that is, older versions of the tree are always preserved.

They have since been used in the Haskell core libraries (in the implementation of *Data.Sequence*), and an implementation in OCaml exists[1] which was derived from a proven-correct Coq specification;[2] and a C# implementation of finger trees (http://dnovatchev.spaces.live.com/blog/cns!44B0A32C2CCF7488!582.entry) was published in 2008; the Yi text editor specializes finger trees to finger strings for efficient storage of buffer text. Finger trees can be implemented with or without lazy evaluation,[3] but laziness allows for simpler implementations.

They were first published in 1977 by Leonidas J. Guibas,[4] and periodically refined since (e.g. a version using AVL trees,[5] non-lazy finger trees, simpler 2-3 finger trees,[6] B-Trees and so on)

See also

- Monoid

References

1. Caml Weekly News (http://alan.petitepomme.net/cwn/2007.10.30.html#5)
2. Matthieu Sozeau :: Dependent Finger Trees in Coq (http://mattam.org/research/russell/fingertrees.en.html)
3. Kaplan, H.; Tarjan, R. E. (1995), "Persistent lists with catenation via recursive slow-down", *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pp. 93–102.
4. Guibas, L. J.; McCreight, E. M.; Plass, M. F.; Roberts, J. R. (1977), "A new

representation for linear lists", *Conference Record of the Ninth Annual ACM Symposium on Theory of Computing*, pp. 49–60.
5. Tsakalidis, A. K. (1985), "AVL-trees for localized search", *Information and Control* **67** (1-3): 173–194, doi:10.1016/S0019-9958(85)80034-6.
6. Hinze, Ralf; Paterson, Ross (2006), "Finger Trees: A Simple General-purpose Data Structure" (PDF), *Journal of Functional Programming* **16** (2): 197–217, doi:10.1017/S0956796805005769.

External links

- http://www.soi.city.ac.uk/~ross/papers/FingerTree.html
- http://hackage.haskell.org/packages/archive/EdisonCore/1.2.1.1/doc/html/Data-Edison-Concrete-FingerTree.html
- Example of 2-3 trees in C# (http://blogs.msdn.com/ericlippert/archive/2008/02/12/immutability-in-c-part-eleven-a-working-double-ended-queue.aspx)
- Example of Hinze/Paterson Finger Trees in Java (http://code.google.com/p/jfingertree/)
- Example of Hinze/Paterson Finger Trees in C# (http://dnovatchev.wordpress.com/2008/07/20/the-swiss-army-knife-of-data-structures-in-c/)
- "Monoids and Finger Trees in Haskell" (http://apfelmus.nfshost.com/monoid-fingertree.html)
- "Finger tree library for Clojure" (https://github.com/clojure/data.finger-tree)
- "Finger tree in Scalaz" (https://github.com/scalaz/scalaz/blob/4bacca9d9aba7c2f0f613c3e5e57d3442b652b21/core/src/main/scala/scalaz/FingerTree.scala)
- "Verified Finger Trees in Isabelle/HOL" (http://afp.sourceforge.net/entries/Finger-Trees.shtml)

Categories: Trees (data structures) | Functional data structures | Algorithms and data structures stubs | Computer science stubs

---