Random Binary Search Trees

A Purely Functional Data Structure

Joe Hurd

Galois, Inc.
joe@gilith.com

Portland Functional Programming Study Group
Monday 13 September 2010

Purely Functional Data Structures

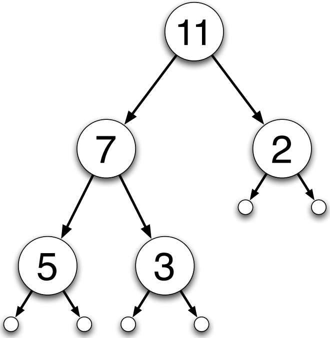
- Purely functional data structures support two operations:
  1. Creating a new object and initializing the data.
  2. Reading the data of an object.
- **Unsupported:** Mutating the data in an object.
  - Simulate mutation by creating a new object that reuses the structure of the old object.
- Benefits:
  - Easy to reason about $\rightsquigarrow$ aggressive compiler optimizations.
  - No thread mutation $\rightsquigarrow$ no concurrency race conditions.
- Drawbacks:
  - Allocation instead of mutation $\rightsquigarrow$ worse performance.

Heaps

A purely functional data structure for finite sets.

- Each node is either a branch or a leaf.
- A leaf is empty.
- A branch contains a key, a left subtree and a right subtree.
- The branch key must be greater than all the keys in its subtrees.

Supports efficient access to the maximum element.

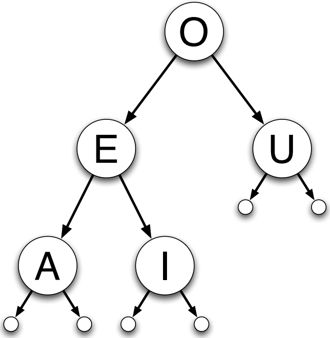


Binary Search Trees

Another purely functional data structure for finite sets.

- Each node is either a branch or a leaf.
- A leaf is empty.
- A branch contains a key, a left subtree and a right subtree.
- The branch key must be greater than all the keys in the left subtree.
- The branch key must be less than all the keys in the right subtree.

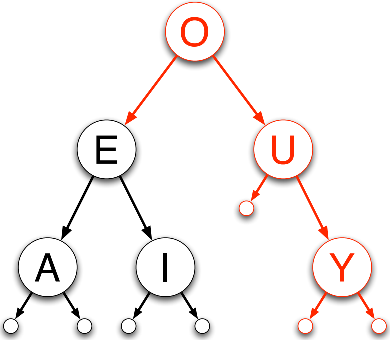
Supports efficient searching for elements.



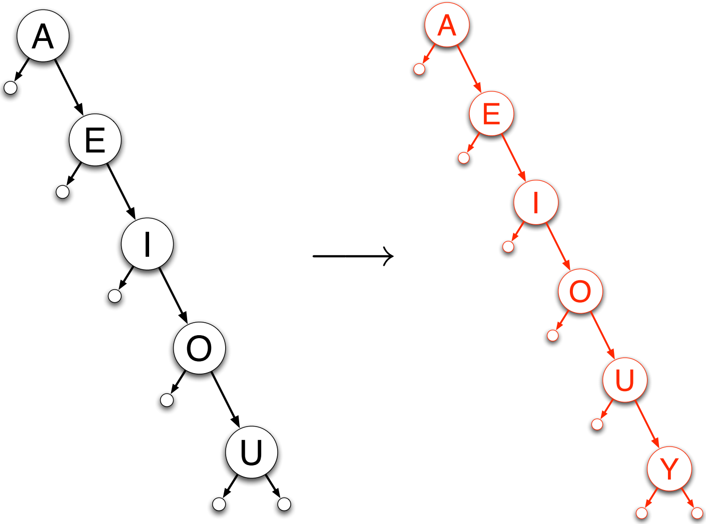
Operating on Binary Search Trees

Must maintain the binary search tree invariants when implementing set operations:

- adding/deleting elements
- union
- intersection
- set difference



Unbalanced Binary Search Trees are Inefficient

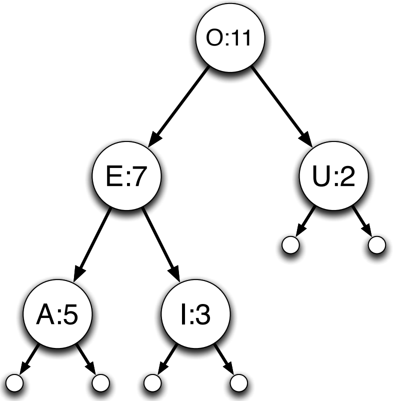


Balancing Strategies

- **In a Nutshell:** Perform additional tree rotations to avoid losing balance.
  - AVL trees [1962]
  - Red/black trees [1972]
  - Splay trees [1985]
- But wait! Most binary search trees are well-balanced.
  - Idea: Given a set of keys, choose a binary search tree containing these keys at random.
  - This will result in good expected performance, independent of the input.

Implementing Random Binary Search Trees

- Given a set of keys with associated priorities, there is a unique binary search tree containing these keys that is also a heap of the priorities.
- Assigning priorities to keys uniformly at random will result in a random binary search tree.
- This idea was told to me by Alistair Turnbull in 2006.



Summary

- Random binary search trees are used to support heavy use of finite sets and maps in formal methods infrastructure.
  1. The Metis theorem prover.
  2. The OpenTheory proof archive.
- I'd like to know how their performance compares with other purely functional data structures for finite sets and maps.
  - Looking for volunteers to carry out experiments...
- The Standard ML code is available under an MIT license from

  `http://src.gilith.com/basic.html`