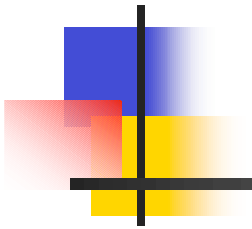
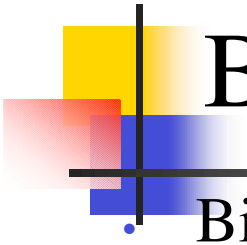


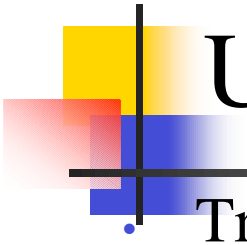
Mutable Trees

Corky Cartwright
Department of Computer Science
Rice University



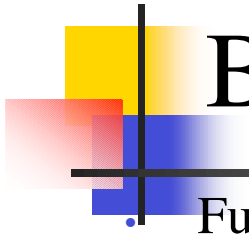
Background

- Binary trees versus general trees
  - Binary tree: every internal node has two subtrees. Internal nodes may or may not contain data. (The former generally preferable.)
  - General tree: each internal node has a list of children. General trees are no longer commonly used as data structures because there is very little fixed structure to support static type checking. We could define a Java data structure GenTree<E> where is the the type of data stored in nodes and then represent arithmetic expressions as GenTree<E> for some appropriate union type E. But this is a bad idea because the typing information is weak (no information about the arity of nodes or type restrictions on subtrees). Justifiably out of favor.
- Binary trees are widely used as a representation sets and maps. Their type structure is fixed so coding with binary trees is a relatively safe process. We will focus on binary trees.



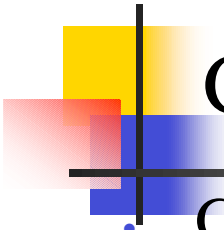
Useful General Tree Concepts

- Tree-walking: in processing trees we visit the nodes in some order. There are three established orders that are named based on when the root of a tree is visited.
  - Pre-order: the root is visited before the children
  - Post-order: the root is visited after the children
  - In-order (applicable only to binary trees) : the root is visited after the left tree but before the right tree.
- Conventional tree-walking has become relatively uncommon because it does not respect type distinctions: roots typically do not have same types as subtrees. It makes sense for input/output and not much else.



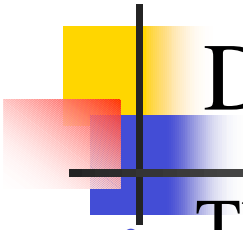
Background on Binary (Search) Trees

- Functional binary trees and composite pattern lists in Java are internally represented using binary tree nodes equipped with `value`, `left` and `right` fields (in some cases additional data fields) and degenerate leaves that either `null` or `false` in Scheme and `null` in Java. Mutable trees have exactly the same fields in internal nodes except that they are mutable.
- In building a mutable search tree, new nodes can be added simply by replacing a degenerate leaf by an internal tree node. The asymptotic cost does not change but the constant factor is reduced.
- **Observation** (independent of OO/Java): trees naturally represent both ordered sets and maps on an ordered set of keys. But these two abstract data structures have incompatible interfaces. Hence, a collections library needs both a TreeSet and a TreeMap.



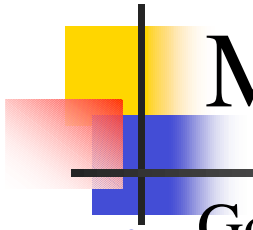
Coding Mutable Tree Data Types

- General Obsevation: treaging boundary conditions (null references) correctly is critical; a frequent source of errors.
- In an OO language, all ugliness should be encapsulated inside the class representing the data structure; the internal representation of a type (*e.g.*, the Node class for a binary tree) should not be visible to clients of the data type.
- Deleting nodes from a binary search tree, no matter what the formulation, is rather ugly, particularly if we are seeking a minimum cost solution. See Cormen et al for code snippets (that are often inscrutable) along these lines. There is a reasonably simple strategy (that was unknown to me until I researched it for this lecture) that is asymptotically optimal that I prefer for pedagogic purposes.



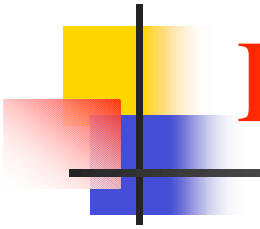
Deleting Nodes from a Binary Search Tree

- This operation is ugly in all formulations of binary search trees, particulary if some notion of optimal code is sought. See Cormen et al for some sample code fragments (they are kinky and clever).
- There is an intelligible strategy with good performance that relies on a cute trick. Deleting the minimum element of a tree is straightforward because the node containing the minimum element has no subtree. General deletion can be reduced to finding the minimum of the right subtree, hoisting it data into the node being "deleted", and deleting the hoisted node.



Mutable Binary Search Trees

- Go to Drjava code



For Next Class

- Laundry homework due next Wednesday. Assignment specs are much longer than the code you must write. Straightforward but not conducive to last-minute solution. Play with it. Have fun. There is nothing conceptually hard about the data or algorithms in this assignment. It is an exercise to help you get up to speed with mutable data structures in Java.
- Two forms for supporting code base:
  - Class per file (prepares you for last two assignments)
  - All classes in one file (easier)
- DrJava makes it easy to practice writing code fragments/exercises. Do it! Don't be afraid to experiment. The interactions pane makes it easy.