CSC207.01 2013F, Class 44: Trees, Generalized

Overview

- Preliminaries.
  - Admin.
  - Questions on the exam.
- Thinking about trees.
- Trees, abstracted.
- A linked implementation.

Preliminaries

Admin

- What did you think about yesterday?
- Fixed unit test for problem 1.
- Upcoming extra credit opportunities:
  - CS Department Talk, Today, Noon (with Pizza), 3821 Writing Bug-Free Code with Theorem Provers
  - CS Table Friday, The New Curriculum
  - Hamlet, Friday (7:30 pm), Saturday (7:30 pm), Sunday (2:00 pm)
  - Swim meet Saturday at some time
  - Typhoon Halyan Relief benefit show, Sunday, November 24th from 7-9pm in Harris. (If the entry fee is a burden, let me know and I'll give you the money.)
  - "Data Sovereignty: The Challenge of Geolocating Data in the Cloud", November 25, 4:15 JRC 101
  - "Gold Fever" by Andrew Sherburne '01 or so, 7:00 p.m., Monday, November 25, ARH 302
  - Tuesday, November 26, 4:15 p.m., JRC 209 a gaming event with the game [d0x3d!]

Questions on the exam

Is the prologue up yet?

> The prologue is now available at http://bit.ly/207exam2pro

Any hints on dealing with the functions as objects problem?

> I'd suggest that you first write the functions assuming that all of the types are integers. Once you've gotten that working, you can start to think about the generic types.

Can you explain the `Iterator.remove` method?

It removes the value you’ve just seen.

Suppose we have the list a b c d and want to remove the b

```
Iterator<...> it = list.iterator();
it.next();  // Returns a
it.next();  // Returns b
it.remove();        // Removes b
```

If we pass all of the unit tests you provide, is our answer correct?

In general, yes. However, you still must take a reasonable approach. For example, you could simulate deletion in BSTs by putting `null` in as the value and then doing some clever maniuplations. But I specify that your really do have to delete nodes and rearrange the tree.

_What should we make anonymous and inner in the iteration problem?

Just the iterator. (You might make the node and cursor inner classes, but it’s not necessary.)

Thinking about trees

- Big idea: A third way to structure data!
  - Chunk of data - Array
  - Linearly Linked nodes
  - Trees: Links go in multiple directions
- Trees can also be ADTs
  - A tree represents relationships between objects
  - Hierarchy at a company
  - Type hierarchies in Java
  - Decision tree
  - Partial order (e.g,. prereqs at Grinnell)

Trees, abstracted

- Philosophy/ Goal
  - Organize items in a hierarchy
- Purpose / Use Cases
  - See above
- Procedures / Methods

Terminology:

- Each item in the tree has zero or more *children*
  - The "arity" of an item is the number of children
- Almost every item in the tree has a *parent*

- One item in the tree is designated as the *root*, which has no parent
- The *depth* of an item is the length of the path from the root to that item
- The *height* of a tree is the maximum depth of any item
- Two items with the same parent are called *siblings*
- The *size* of a tree is the number of items
- A *leaf* is an item with no children.

What methods should we provide? (assume we're trying to model hierarchies, not decision trees or partial orders)

- Observers
    - Tree.depth(item)
        - Might be a "node"
        - Might be a "location"
        - Might just be the name of a value
    - arity(item) - How many classes have 151 as a direct prerequisite
    - height() - Get the height of a tree
        - Implementation one: Recurse through the tree O(n)
        - Implementation two: Store it as a field in each node (assuming we're using nodes)
        - Implementation three: Store it in the tree
    - leafp(item) - Is it a leaf?
    - size()
        - Implementation one: Recurse through the tree O(n)
        - Implementation two: Store it as a field in each node (assuming we're using nodes)
        - Implementation three: Store it in the tree
    - int sibs(item)
        - How many siblings?
        - Or maybe an iterator
        - Or maybe an array
        - Or maybe ...
        - If siblings are ordered leftSib rightSib
        - Item parent(Item item)
        - sib?(Item me, Item you)
        - children(Item item)
            - Iterator?
            - Array
        - get(Item item)
            - Necessary if we distinguish nodes/locations from values
    - Iterator leaves()
- Mutators

An array-based implementation of binary trees

- Each value we store gets an index.
- Store in "breadth-first" order leave blanks for missing nodes and missing children
- The children of the value at position i are at ...?
- The parent of the value at positin i is at ...?

Here’s a tree of the indices we’d get

```
               0
            /      \
          1          2
        /   \      /   \
      3      4    5      6
     / \    / \  / \    / \
    7   8  9  10 11 12 13 14
```