Lecture 17: Expression Trees & Array Representation of Trees

CS 62
Spring 2012
Kim Bruce & Kevin Coogan

Representing Expressions

- Represent $3 \times 7 + 6 / 2 - (3 + 7)$ as tree
  - Parser builds tree
  - Send message to tree to print or evaluate
- Mutual recursion in parser
- Different classes for different kinds of nodes.
- See Parser code

Parsing Expressions

Array Representations of Trees
Array Representation

- data[0..n-1] can hold values in trees
  - left subtree of node i in 2*i+1, right in 2*i+2,
  - parent in (i-1)/2

Indices: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
data[]: U O R C M E S -- -- P T -- --

Min-Heap

- Min-Heap H is complete binary tree s.t.
  - H is empty, or
  - Both of the following hold:
    - The value in root position is smallest value in H
    - The left and right subtrees of H are also heaps.
      Equivalent to saying parent ≤ both left and right children.
  - Excellent implementation for priority queue
  - Dequeue elements w/lowest priority values before higher

Array Representation: Efficiency

- Tree of height h, takes $2^h-1$ slots, even if only has $O(h)$ elements
  - Bad for long, skinny trees
  - Good for full or complete trees.
- Recall complete tree is full except possibly bottom level and has all leaves at that level in leftmost positions.

PriorityQueue

```
public interface PriorityQueue<E extends Comparable<E>>
{
    /**
     * @pre !isEmpty()
     * @return The minimum value in the queue.
     */
    public E remove();
    public E getFirst();
    public void add(E value);
    public boolean isEmpty();
    public int size();
    public void clear();
}
```
Implementations

- As regular queue (array or linked) where either keep in order or search for lowest to remove:
  - One of add or remove will be $O(n)$

- Heap representation (in arraylist) is more efficient: $O(\log n)$ for both add and remove.
  - Insert into heap:
    - Place in next free position,
    - "Percolate" it up.
  - Delete:
    - remove root,
    - move smallest child up to fill gaps, repeat

Insert 15:

IndexRange: 0 1 2 3 4 5 6 7 8 9 10
data: 10 20 14 31 40 45 60 32 33 47

IndexRange: 0 1 2 3 4 5 6 7 8 9 10
data: 10 20 14 31 40 45 60 32 33 47 15

IndexRange: 0 1 2 3 4 5 6 7 8 9 10
data: 10 15 14 31 20 45 60 32 33 47 40