Binary Search & Splay Trees

CS 62 - Spring 2015
Michael Bannister

Assignment

- Work first on World & Species classes
  - Need JUnit for both
  - Moves hop, left, right, and infect use up turn
  - if's and goto are free

Min-Heap

A **min heap** is a complete full binary tree such that, the value in the root position is the smallest value and left and right subtrees of the root are both min heaps.

Typically a min heap is stored in an array using the array representation of trees.

See VectorHeap code

Heap Performance

- remove: O(log n)
- getFirst: O(1)
- add: (log n)
- isEmpty, size, clear: O(1)

Heapsort

1. Add all elements from vector into heap: O(n log n)
2. Remove elements in order from heap: O(n log n)
3. Total time: O(n log n)

Can get about a factor of 2 speed up by building the heap “in place” in O(n) time using heapify.

Binary Search Tree

A **binary search tree** is a binary tree that is either empty or every value in its left subtree is less than (or equal to) the value of the root and every value inits right subtree is greater than the value of the root.

Used as a more efficient sorted list!

BST Interface*

```
public class BinarySearchTree<E extends Comparable<E>> {
 // Public methods
 public void add(E value) // O(log n)
 public E remove(E value) // O(log n)
 public boolean contains(E value) // O(log n)
 public E get(E value) // O(log n)
 public boolean isEmpty() // O(1)
 public void clear() // O(1)
 public int size() // O(1

 // Important helpers
 protected BinaryTree<E> locate(BinaryTree<E> root, E value)
 protected BinaryTree<E> predecessor(BinaryTree<E> root)
 protected BinaryTree<E> successor(BinaryTree<E> root)
 protected BinaryTree<E> removeTop(BinaryTree<E> topNode)
}
```

Locate Value

- If tree is empty, then not found!
- If **value** is at root, then found!
- If **value** ≤ root, call locate on left subtree.
- If **value** > root, call locate on right subtree

When **value** is not in the tree, locate finds where it "should" be!

Predecessor and Successor

- Predecessor of root
  - In left subtree go right as far as possible
- Successor of root
  - In right subtree go left as far as possible

Add Value

- Use locate to find position of where the **value** should be, and insert node with **value**.
- If **value** is already in the tree, then more work is needed to adjust tree. See code and example on the board.