CSCE 156 Lab: Binary Search Trees & Heaps

Handout

0. Prior to the Laboratory

1. Review the laboratory handout
2. Read and understand relevant materials on Binary Search Trees and Heaps

1. Lab Objectives & Topics

Upon completion of this lab you should be able to:

- Be familiar with Binary Search Trees and Heaps
- Be able to implement and utilize tree traversal algorithms
- Be able to utilize BSTs and Heaps in an application

2. Problem Statement

Activity 1: Binary Search Tree

Recall that a Binary Search Tree (BST) is a data structure such that elements are stored in tree nodes. Each tree node has a pointer (reference) to its parent, left child, and right child. Each node in a BST also has a *key* associated with it. The Binary Search Tree Property is such that:

1. All keys in a node's left-subtree are *less* than that node's key
2. All keys in a node's right-subtree are *greater* than that node's key

A binary search tree Java implementation has been provided for you. It has basic functionality to add, remove, and retrieve elements. Each of these operations maintains the BST properties. Your activities will include adding additional methods to traverse the tree using several different traversal strategies and to count the number of leaves in the tree.

A node in a binary tree is called a *leaf* if it has no children. Since the structure of a binary search tree is determined by the order elements are entered, there is no easily computable formula for the number of leaves. Instead, a BST needs to be traversed and the leaves counted up.

There are three traversal strategies that you will implement. The implementation is up to you; you may find that a recursive strategy is the quickest to implement or you may find a Stack data structure useful. Each of these strategies begins at the root and visits a node and its children in different orders.

1. Preorder Traversal: visits the node, then visits the nodes in the left-subtree then in the right-subtree
2. Inorder Traversal: visits the left-subtree first, then the node itself, then the right-subtree
3. Postorder Traversal: visits the left-subtree first, then the right-subtree, then the node itself

Instructions

1. Download the Eclipse project file available on Blackboard
2. Implement the three ordering methods as described above
3. Implement the `getNumLeaves()` method
4. Use these methods to answer the questions on your worksheet

Activity 2: Heaps

A heap is a data structure similar to a binary search tree in that it's structure is a binary tree. The primary difference is that the key for each node in the tree is larger than *both* of its children. There is no restriction on the relation between the keys of child nodes. Another condition of heaps is that they are *full* binary trees: each level of the tree, with the possible exception of the final row has every node present. This property ensures that operations such as add and remove (from the top) can be performed with O(log(n)) operations.

Implementing a heap is usually done using a dynamic array—the random access ensures the optimal behavior of operations. For this activity, we will not be building a heap from scratch; rather we will be taking advantage of the Java Collection's library `PriorityQueue` class. This class uses a comparator provided at instantiation to maintain the following guarantee: any dequeue operation (`poll`) will return the element with the "highest" priority (the greatest value according to the comparator). The corresponding enqueue operation (`offer`) inserts elements into the priority queue.

One application of heaps is in the *heap sort* algorithm. The basic idea is that elements are removed from a list one-by-one and are placed on a heap. Then, elements are successively removed from the heap and placed back into the list. The heap property guarantees that the maximal (or minimal) element is removed each time, imposing an ordering.

Instructions

1. Implement the `Heap` class as specified; each method should be implemented using methods of the underlying `PriorityQueue`.
2. Use this implementation to implement the `heapSort` method in the `HeapSort` class. This class has a main method which you can use to test your implementation.

Advanced Activity (Optional)

Another tree traversal strategy is called a Breadth-First-Search traversal strategy. Starting at the root, nodes are visited level-by-level in a left-to-right order. Design (or research) and implement a Breadth-First-Search strategy on your binary search tree. Hint: look at the non-recursive traversal strategies used in some of the methods provided to you. Think of a similar strategy using a different data structure.