COMP 272

Data Structures II

Midterm-Exam

Due Tuesday Oct 19th by 2pm on Sakai

No late submissions will be accepted. Try to submit by 1:55 pm so that your submission will be accepted if Sakai clock slightly differs from yours. Make sure you submit the right files. Explanations such as internet connection failure, or computer breakdown, or not able to access vm are not acceptable explanations for late submissions. Answer all questions.

This would count for 75% of the midterm grade. Follow the honor code by not discussing the solution with any person by any means, directly or indirectly. You can consult online resources, books, etc., but not allowed to post questions in online forums or any type of electronic communication. Academic dishonesty will result in a score of 0 for the entire midterm. I will be happy to answer anything you may want to seek clarification on.

1. In the MyLinkedList.java create Java methods to perform the following operations.

(i) public MyLinkedList<E> uniqueList() {
}

returns a new MyLinkedList<E> that takes this MyLinkedList<E> and removes all repetitions of any value. E.g., {1,2,3,3,4,2,1,5} list will return {1,2,3,4,5} when E is Integer. You can assume existence of equals() method. The original list should be unaltered.

(ii) public MyLinkedList<E> commonValues (MyLinkedList<E> other) {
}

that takes this MyLinkedList and the other MyLInkedList and returns a MyLinkedList containing values that are common to both lists with no repetitions. For example, given, this List is ={1,2,3,3,4,2,1,5} and other List ={1,5,5,2,10,10,11}, method *commonValues*() will return a new List {1,2,5}. If they have nothing in common, return an empty List.

(iii) public MyLinkedList<E> union (MyLinkedList<E> other) {
}

takes this List and the other List and returns a list containing everything from both lists without repetitions. For example, given the two lists above, method *union*() will return a new List {1,2,3,4,5,10,11}. If both are empty then return empty, if one is empty, return the other eliminating repetitions. The original Lists should be unaltered after the method call.

2. In the MyBigInteger.java create Java methods to perform the following operations.

(i) public MyBigInteger multiply (MyBigInteger other){
}

Takes *this* MyBigInteger (of any size *n*) and the other MyBigInteger (which contains a maximum of two digits or size<=2) and returns the product of the two numbers as a MyBigInteger. Assume that MyBigInteger is always a non-negative integer. State the time-complexity in big-oh notation as a function of *n.*

(ii) public boolean even() {
}
Returns true if this MyBigInteger is even, and false if it is odd.

3. In the BinaryTree.java create Java methods to perform the following functions:

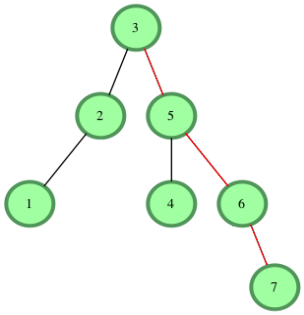
(i) public int height() {
}
Calculates and returns the height of *this* BinaryTree. Height of a tree with only root node is 0.

(ii) public boolean isLeaf(Node<E> v) {
}
Returns true if node v is a leaf and false if it is not a leaf. If v is null throw exception.

(iii) public int nodeLevel(Node<E> v) {
}
Calculates and returns the level number of a specific node in the BinaryTree. The level number of root is 0. The level number of each child of the root is 1 and so on. If v is null throw exception.

(iv) public void breadthFirstTraversal() {
}
Prints each node info (label) in a breadth-first traversal of the binary Tree starting from the root, for a non-empty tree. If empty, throw an exception.

(v) public int nodeHeight(Node<E> v){
}
Calculates and returns the height of a node in the binary tree. The height of a leaf node is 0. The height of any node p in the binary tree is the number of edges from the node to the farthest leaf node in the subtree rooted at p. Example: In the tree below, height of node 5 is 2. Height of the root is 3. Height of node 2 is 1. Height of node 7 is 0.



(vi) public double averageLevel(){
}
Calculates and returns the average Level of the binary tree. This is the sum of the level number of every node of the tree divided by the number of nodes. For the binary tree above, the averageLevel= (0+1+1+2+2+2+3)/7 = 11/7=1.57

4. Extend BinaryTree.java as follows
public class BinaryTreeVerification<E extends Comparable<E>> extends BinaryTree<E> {
} to include the following methods

(i) public boolean isBST() {
}

Returns true if this binary tree is a binary search tree and false if it is not.

(ii) public boolean isFull() {
}
Returns true if this binary tree is a full binary tree and false if it is not.

(iii) public boolean isComplete() {
}
Returns true if this binary tree is a complete binary tree and false if it is not.

5. In BinarySearchTree.java write a method with the following functionality.
   public MaxHeap<E> buildHeapFromBST() {
   }
   This method takes *this* BinarySearchTree and using MaxHeap.java code and its methods, produces a max heap based on the contents of the binary search tree and returns it.

6. No code needed. Using pictures show the result of inserting 2, 1, 4, 5, 9, 3, 6, 7 into an initially empty AVL tree, one value at a time. Indicate what type of rotation needed if any.