End Term Evaluation – Data Structures

For this assignment, you will solve problems based on what you have learned in Data Structures.

Instructions

- Review notes of the Chapter.
- There are 3 questions in this assignment.
- **Assignment submitted after due date will not be evaluated and a score of zero will be awarded for this assignment.**
- Combine all files into **one zip file**, which you submit via email.

Due Date: Midnight, July 01, 2020.

Questions:

Ques. 1 In this problem, you have to build a simple binary search tree. Do remember, this tree won't handle duplicate values as they are more complicated and result in trees that are much harder to balance. Be sure to always remove duplicate values or check for an existing value before inserting.

1. Define a **Node** structure . It should have attributes for the data it stores as well as its left and right children. As a bonus, try including the Comparable module and make nodes compare using their data attribute.

2. Define a **Tree()** function which accepts an array when initialized. The Tree function should have a root pointer varaible which uses the returned value of build_tree which you'll write next.

3. Define a **build_tree()** function which takes an array of data that you have accepted inside Tree() function (e.g. [1, 7, 4, 23, 8, 9, 4, 3, 5, 7, 9, 67, 6345, 324]) and turns it into a balanced binary tree full of Nodes appropriately placed (don't forget to sort and remove duplicates!). The build_tree function should return the pointer to the root node.

4. Define a **insert()** and **delete()** function which accepts a value to insert/delete (you'll have to deal with several cases for delete such as when a node has children or not).

5. Define a **find()** function which accepts a value and returns a pointer to the node with the given value.

6. Define a **level_order()** function. The method should traverse the tree in breadth-first level order and display the data stored in each node. This function can be

implemented using either iteration or recursion (try implementing both!). As a bonus, make the function return an array of values. **Tip:** You will want to use an array acting as a queue to keep track of all the child nodes that you have yet to traverse and to add new ones to the list.

7. Define **inorder(), preorder(), and postorder()** functions. Each function should traverse the tree in their respective depth-first order and display the data stored in each node. As a bonus, make the function return an array of values.

8. Define a **depth()** function which accepts a node and returns the depth (number of levels) beneath the node.

9. Define a **balanced()** function which checks if the tree is balanced. A balanced tree is one where the difference between heights of left subtree and right subtree is not more than 1.

10. Define a **rebalance()** function which rebalances an unbalanced tree. **Tip:** You'll want to create a level-order array of the tree before passing the array back into the **build_tree()** function.

11. Write a simple driver script that does the following:

1. Create a binary search tree from an array of random numbers (make use of rand() function)
2. Confirm that the tree is balanced by calling '**balanced'**
3. Print out all elements in level, pre, post, and in order
4. Try to unbalance the tree by adding several numbers > 100
5. Confirm that the tree is unbalanced by calling **'balanced'**
6. Balance the tree by calling **'rebalance'**
7. Confirm that the tree is balanced by calling **'balanced'**
8. Print out all elements in level, pre, post, and in order

Ques. 2 This problem uses **Command Line Arguments, File Handling, and the Implementation of Queues using (circular) arrays**

Write a Program, which takes as input one filename, and processes the Queue Insert/Delete Commands written in the file.
i.e. User will run the program like
./a.out file1

File format of file1
Insert 10
Insert 20
Remove
Remove
Remove

Expected Output
Inserting 10.
Queue is : 10
Inserting 20.
Queue is : 10 20
Remove returned 10
Remove returned 20
Remove FAILED due to queue empty

Implementation Hints
You can use strcmp() to do comparison of strings.
File Reading logic can be
fscanf(fp, "%s", command)
if (!strcmp(command, "Insert"))
  fscanf(fp,"%d", &intvalue)

Ques. 3 A bag (also called multiset) is a generalization of a set, where elements are allowed to appear more than once. For example, the bag {a, a, b} consists of two copies of a and one copy of b. However, a bag is still unordered, so the bags {a, b, a} and {a, a, b} are equivalent.

In this task you have to implement some features for a linked representation of bags. This representation is very similar to a regular singly-linked list, except for the following:

In addition to the value and the reference to the next cell, each bag cell stores the number of copies of its value (see Figure 1), which is always positive.

For a given value, at most one cell storing that value should appear in the data structure.

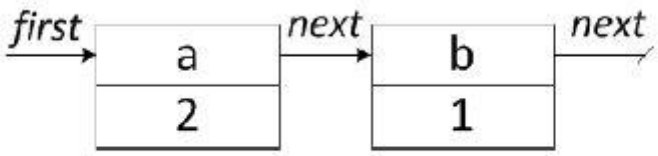

Figure 1: A possible linked representation of the bag $\{a, a, b\}$.

Implement the following functions:
add (v: G; n: INTEGER), which adds n copies of v to the bag.

remove (v: G; n: INTEGER), which removes as many copies of v as possible, up to n. For example, removing one copy of 'a' from the bag {a, a, b} will result in a bag {a, b}, while removing two copies of 'c' from the same bag will not change it.

subtract (other: LINKED BAG [G]), which removes all elements of other from the current bag. For example, taking the bag { a, a, b} and subtracting { a, b, c} from it will yield the bag { a }.

Display( ), which displays all the values along with their frequencies at any given point of time.

Your implementation should satisfy the provided constraints.

****